MSX FAN SERIES

3

マシン語入門 実践編

マシン語活用のための実践テクニック

MSX FAN SERIES 3 マシン語入門(実践編)

MIA

MSX PAN SERIES

マシン語入門 実践編

まえがき

「マシン語の勉強を始めたけど，さっぱりわからないわ」「こんな細かい命令，どれをどうつないだらプログラムになるんだ？」「マシン語なんて，オレにわかるわけないや」…こんなことを思ってるあなた，その**悩み**は健全な悩みです。

変化の激しいこの世の中，マシン語のひとつやふたつできないでどうする！　とまではいいませんが，MSXの世界を**極めよう**と思ったら，マシン語を知ってるのが一番。ところが，「マシン語は難しい」などというヘンな話が世の中に広まって，マシン語は大変迷惑しています。中には，そんなウワサを真にうけて，マシン語の世界の扉の前で立ち止まってしまう人までいるのですから。でもそんなウワサ，僕らはぜんぜん信じていません。

ほんとうは。マシン語はぜんぜんむずかしくないのです。ただ**ヤヤコシイ**だけ。「ヤヤコシイ」のと「むずかしい」のはよく似ているようでいて，実は微妙にちがっています。からんだ糸をほぐすときのように，気持ちをしっかり持って根気よくやれば必ずわかるもの。何よりマシン語は**実践**が大事。糸のほぐしかたは，何度もほぐしてるうちに身につくものなのです。

それに。マシン語というモノが「ヤヤコシイ」ことであるがため，先輩たちが残した**テクニック**がよけいに必要になってきます。それを系統だてて教えてくれる本はなかなかないのですが，この本は無理や誤解をかえりみず，それに挑戦しています。

「暴走させてしまった…」が肌身にしみる言葉になったとき，あなたももうマシン語フリークの仲間入りです。この本を読んだあなたがゆくゆくは**MSX**のマシン語**エキスパート**になってくれることを，僕らは望んでいます。

1984年　6月　　渡辺卓也，樋口賢治

目次

1章 これだけは知っておこう

BASICを卒業し，そろそろマシン語をと思ってメモリや16進数の解説を読んでみたがどうもピンとこない。レジスタを操作するような細かい命令をどうつなげたら意味のある動作になるのか，さっぱりわからない。複雑な動作の命令を実習しながら理解したい。マシン語プログラムをつくるちょっとしたきっかけになるようなものはないだろうか。このような人々にぴったりな実践的基礎知識集。

本書でのリストの見かた

本書の内容に入ってゆく前に，この本に掲載されているリストの見かた，入力法・実行法などについて少々触れておきましょう。

本書は小社発行のMSX用「モニタ・アセンブラ」を使ってプログラムを作成，アセンブルしています。掲載したリストはすべて「アセンブル・リスト」と呼ばれる形式です。

図1—1 アセンブル・リストの形式

```
1000:    D000                   ORG  0D000H
1010:                           ;
1020:    D000     AF            XOR  A
1030:    D001     210010        LD   HL, 1000H
  ①       ②        ③               ④
```

図1-1を見てください。ここで，①はエディタでソース・プログラムを書いたときについた行番号です。このアセンブラはMSX-BASICのエディタ部を借用してソース・プログラムを編集するため，BASICと同じように行番号がついています。

②はアセンブルによって生成されたオブジェクト（マシン語のコード)がどの番地に格納されるかを示す16進数です。この例では，“XOR A”というニモニックをアセンブルしてできた“AF”というオブジェクト・コードが“D000”番地にしまわれることを示しています。③がオブジェクトを表示している欄です。オブジェクトがどのメモリ・エリアに格納されるかを決めているのが先頭の“ORG 0D000H”という部分で，ここを書きかえれば“AF”以降はC000H番地からでも2000H番地からでも，格納場所を自由に変更できるのです(その番地が自由にオブジェクトを格納できるエリア内かどうかは別問題)。

④はニモニックを表示する欄です。1010行の；(セミコロン)はその行がコメント行(BASICでいうREM文)であるということです。当然この行はオブジェクトがなく，②，③とも空欄になっています。

この図では省略されていますが，本書で作成したプログラムは最後を“RET”で終わるようになっています。つまり「サブルーチン」の形式になっているということです。このことはのちに詳しく説明しますから，とりあえず最後にRETをつけなければならないということだけ覚えておいてください。

また，「モニタ・アセンブラ」をロード，実行した直後，必ずダイレクトに“CLEAR 100，& HD300⏎”を実行してください。

さて，手持ちのMSXにこれらのプログラムを入力するときにはどうすればよいのでしょうか。その方法には2通りあります。すなわち，

①ソース・プログラム(ニモニック)を打ち込んで，アセンブルする方法。

②モニタ・プログラムでオブジェクト(③の部分)だけを入力する方法。

このうち①は，ソース・プログラムを手元に残しておける(すなわち自由に改造できる)という点ですぐれていますが，何せ手間がかかります。②は比較的楽に入力できますが，拡張性に欠け，誤りを発見できにくいところが難点です。どちらでも選べるようにアセンブル・リストで掲載したのですが，なるべく①の方法で入力してください。ソース・プログラムが残っているということは大きな強みです。自由に書きかえて，いろいろと実験を繰り返し，数えきれないほどの失敗をしてみることが上達への早道です。面倒なのは慣れるまで。ソースの入力に慣れるころには，ニモニックのほとんどが親しいものになっているでしょう。

「モニタ・アセンブラ」を持っていない人は入力する方法がありません。②の番地に③のマシン語を入力するためには何らかのモニタ・プログラムが必要だからです。モニタ・プログラムなくしてマシン語を勉強しようというのはちょっと無理な注文です。さっそく小社のパッケージ「モニタ・アセンブラ」(グラフィック・エディタ付で4,800円)を買いましょう。本書の兄貴分の「MSX fanシリーズ①マシン語入門・基礎編」(1,800円)には全リストが公開されていますから「モニタ・アセンブラ」を入力する手間をいとわない人は，これでもいいですね。

とにかく，「モニタ・アセンブラ」を持って，アセンブル・リストの見かたを理解すれば，あとはMSXでマシン語プログラムを作ってみようという意気込みだけで，これからの内容は納得いくものになることでしょう。

1 これだけは知っておこう
何はともあれプログラムしてみよう

「マシン語で何かプログラムを書いてみよう」と思い立った動機はさまざまでしょう。BASICに飽きた。BASICでは遅い。BASICの次はマシン語を勉強したい。などなど。

そんなあなたが，今いちばん心配しているのは「やはりマシン語はBASICより難しいんだろうな」ということではないでしょうか。しかしここで私たちは断言します。「マシン語は難しくない。ただ複雑なだけだ」と。

「マシン語入門・基礎編」で説明されているとおり，マシン語は１命令の動作で影響を及ぼせる範囲が狭く，BASICの１命令でできる操作がマシン語の10命令以上になることさえあります。ということは，マシン語である動作をするときは，場合によって命令数がBASICの10倍以上になることもあるということです。命令数が増せば当然手間もかかりますし，誤りの入る余地も多くなりますね。ここが「マシン語は難しい」との風評がたつゆえんなのです。

手間がかかることと並んでマシン語が難しいといわれる理由にあげられるものがもう１つあります。それはマシン語はBASICのように「管理されていない」ということです。ごぞんじのように，BASICのプログラムは“Run”という命令を与えれば実行を始めます。ここで行番号を見ながら実行を制御しているのは誰でしょう？　エラーが起きたところで実行を停止させているのは？　そもそもエラーの発生を検知しているのはどこの誰でしょうか!?　そういう疑問を感じたことはありませんか。もしこれまで疑ったことがなかったとしても，これからマシン語を実習してゆくうちにこのことをいやというほど思い知らされるにちがいありません。

BASICは「インタープリタ」と呼ばれるものが管理・実行しています(インタープリタ自身はマシン語で書かれています)。対するマシン語は誰も管理していません。マシン語を実際に解釈する“CPU”と呼ばれる頭脳は，指示された番地の内容をマシン・コードと見なして高速に実行するだけで，ここにこの命令がくるのはおかしいとか，こんなところにジャンプするはずがないとか，数値があふれたから実行を停止しようとかの疑問は一切もちません。いわば想像をはるかに超えた「バ

カ正直」といったところです。

CPUが人間の予定を外れた動作をし，それを止めるおだやかな手立てが見つからないとき「暴走した」といいます。このことについては後に詳しく書きますが，とりあえず，「マシン語はいつも野放しで実行されている」ということは覚えておいてください。

以上，マシン語がBASICなどの高級言語よりも難しいといわれる理由をあげました。ここでまとめておきましょう。

①マシン語は１つ１つの命令の動作範囲が狭いため，ある仕事をさせようとすると高級言語よりも多くの命令数を要する。すなわち面倒。
②命令数(作業)が増えるにつれて，人間が誤る可能性が多くなる。
③いったん実行を始めたマシン語は，誰も管理できない野放し状態である。

この扱いにくいマシン語を自由に繰るためには，まず機械(CPU)のように正確な論理を追跡しなければなりません。また，取り扱う事がらを基本的な小事項に分割してマシン語命令のかたまりに置きかえなければなりません。根気も必要です。そしてもちろん，マシン語の命令を使うコツに精通していなくてはなりません。

これらは経験を積めば到達できることです。誰にもできます。まずやってみることです。１つ１つの小さな命令をどう組み合わせれば意味のある動作になるか，これから実際に動かしながら見てゆきましょう。

1.メモリの内容をコピーしよう

さあ，マシン語入門の実践編，手始めはメモリの操作からです。操作といっても話は簡単，ただある番地の内容を別の番地にコピーするだけです。

MSXに使われているCPU・Z80には，メモリの内容を各種のレジスタにもってくる命令が多く準備されています。手近にあるZ80の命令表のロード命令の欄を見てください。オペランドにカッコつきのものがある命令は，メモリの内容を出し入れする命令です。

しかし，よく見てください。ロード命令は２つのオペランドを持っていますが，２つともカッコつきである命令はありませんよね。つまり，

```
LD (1000H), (2000H)
```

などという操作は１命令でまかなえないわけです。そこで，これと同じ動作をする命令群を考えてみましょう。MSXのRAMエリアは最低の16KシステムでC000H番地からFFFFH番地までで，「モニタ・アセンブラ」をロードするとC000H番地からD4FFH番地までに制限され

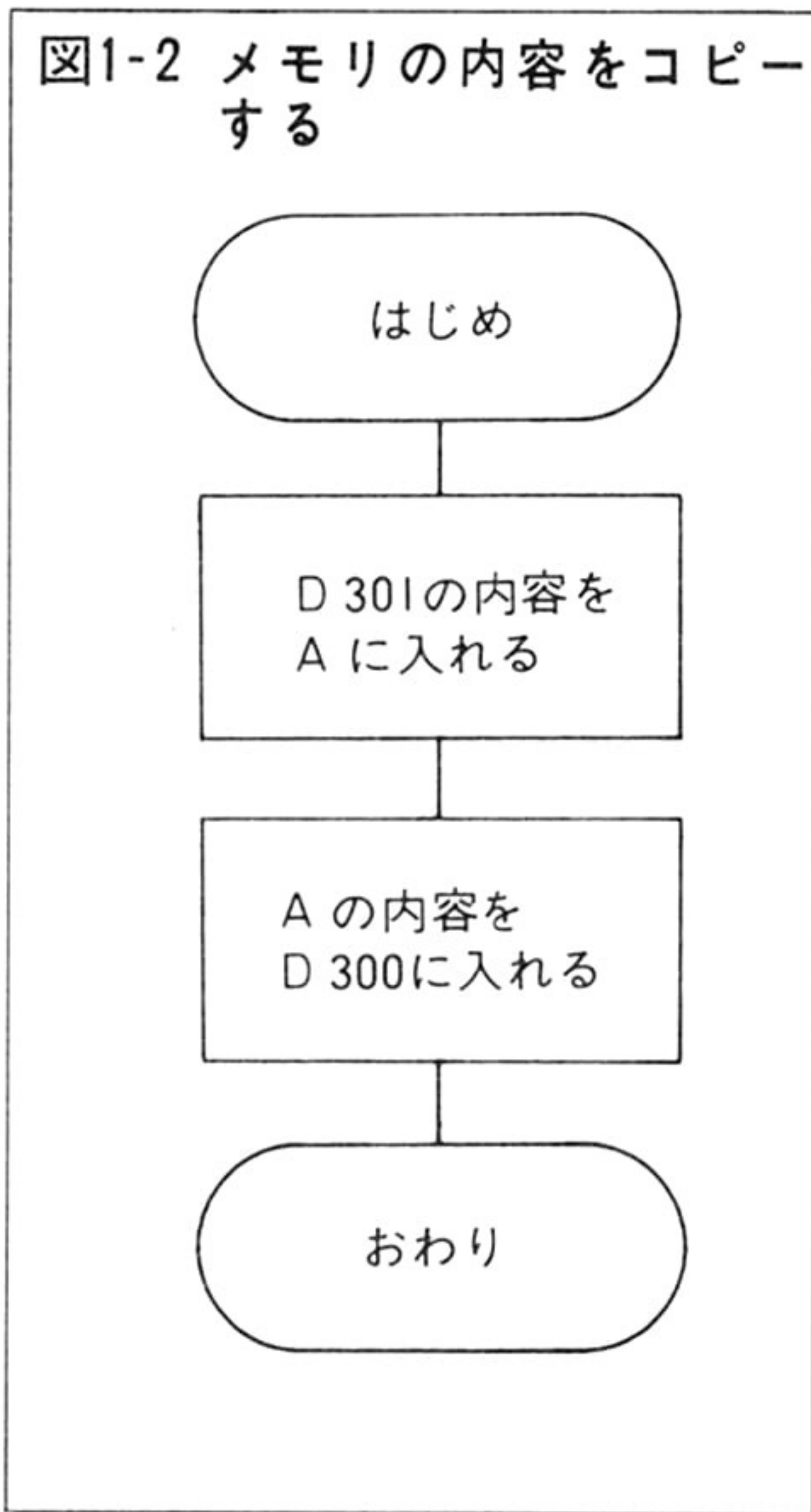
ます。ここでは，

LD (D300H), (D301H)

を例にしてみます。

まず処理の順を明らかにするために，フローチャートを書いてみましょう。**図1-2**のようになりました。使うレジスタは必ずしもAである必要はないのですが，ここではAにしておきます。

フローチャートをもとにして書いたプログラムが**リスト1-1**です。たった7バイト，D400H番地～D406H番地までです。「モニタ・アセンブラ」を実行したら，CLEAR100，&HD300として**リスト1-1**のニモニックを(行番号をもちろんつけて)入力し，CMD ASMでアセンブルしてください。エラーなくアセンブルを終了したら，CMD MONでモニタを呼び出し，Rコマンドでオブジェクトをメモリに格納します。

図1-2 メモリの内容をコピーする

リスト1-1

```
MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                    ORG      0D400H
110:    D400 3A01D3             LD       A,(0D301H)
120:    D403 3200D3             LD       (0D300H),A
130:    D406 C9                 RET
```

さて，作ったプログラムを実行する前にモニタのSコマンドでコピーされるデータをあらかじめメモリにセットしておきましょう。SD301↵としてみてください。コピーされるデータはわかりやすいように50Hとしましょう。コピー先のD300H番地が偶然50Hでないかどうか，Dコマンドで確認することも忘れずに。DD300，D301↵とすると**リスト1-2**のようになっているはずです。

リスト1-2

```
*DD300,D301
D300 △△  50        △△の部分は何が表示されてもかまわない。
```

さあ，実行です。モニタでGD400⏎としてみましょう。まるで何もしなかったかのごとく，一瞬にしてBASICのコマンド待ちに戻りますね。もう1度CMD MON ⏎としてからDコマンドで，ほんとうに何かをしたかどうかを確認してみてください。D300H番地の内容が50Hになっていれば大成功です。

これがメモリの内容をコピーする方法です。リストを見るとあっけないでしょう？ ここでは1バイトだけをコピーしましたが，複数バイトも基本は同じです。繰り返しを使う場合の方法は，のちに書きます。

2.画面に文字を表示してみよう

次に，画面に文字を表示してみます。

MSXでマシン語のプログラムをつくるとき，主に入出力に関する部分はBIOS(Basic Input Output System)と呼ばれるものを使うと便利です。BIOSはサブルーチンの集合で，BASICインタープリタといっしょにMSXのROMにしまわれています。ユーザーはBIOSの使いかたと使う場所にさえ気をつければ，BIOSの中身を知る必要がありません。これもMSXの互換性を保つ1つの工夫ですね。

BIOSを呼び出す番地を「エントリーアドレス」といいます。主要なBIOSのエントリーアドレスと機能を**表1-1**に示します。これで全部ではありませんが，これだけ知っていれば今のところは十分でしょう。全部を知りたいときは，巻末の参考文献(1)～(2)を参照してください。BIOSはBASIC ROM内にあるのでROMOSとも呼ばれます。

表1-1　主なBIOS

番　地	ルーチン名	説　明
009F	CHGET	入力があるまで待ち，入力した文字コードを求めます。 ◇エントリーパラメータ　：なし ◇リターンパラメータ　：Aレジスタに入力した文字コードが入ります。 ◇使用レジスタ　：AF
00A2	CHPUT	画面に，指定した文字を表示します ◇エントリーパラメータ　：Aレジスタに文字コードを入れます。 カーソルX-F3DD：Y-F3DC ◇リターンパラメータ　：なし ◇使用レジスタ　：なし
00A5	LPTOUT	プリンターに，指定した文字を出力します。 ◇エントリーパラメータ　：Aレジスタに文字を入れます。 ◇リターンパラメータ　：異常終了したらキャリーフラグをたてます。 ◇使用レジスタ　：F
00AE	PINLIN	キャリッジリターンやSTOPを入力するまでに入れた値をラインバッファに入れます。 ◇エントリーパラメータ　：なし ◇リターンパラメータ　：HLレジスタにラインバッファの先頭番地－1が入ります。 STOPを入力したときのみ，キャリーフラグがたちます。 ◇使用レジスタ　：すべて

番地	ルーチン名	説明
00B7	BREAKX	Control-STOPを押したかどうか調べます。 ◇エントリーパラメータ：なし ◇リターンパラメータ　：押していたらキャリーフラグがたちます。 ◇使用レジスタ　　　　：AF
00C0	BEEP	ブザーをならします。 ◇エントリーパラメータ：なし ◇リターンパラメータ　：なし ◇使用レジスタ　　　　：すべて
00C3	CLS	画面をクリアします。 ◇エントリーパラメータ：ZフラグがONならクリアします。 ◇リターンパラメータ　：なし ◇使用レジスタ　　　　：AF、BC、DE
00C6	POSIT	指定した位置にカーソルを移動します。 ◇エントリーパラメータ：Hレジスタにカーソルの水平位置を，Lレジスタにカーソルの垂直位置を指定します。 ◇リターンパラメータ　：なし ◇使用レジスタ　　　　：AF
00D5	GTSTCK	ジョイスティックの状態を調べます。 ◇エントリーパラメータ：Aレジスタに調べるジョイスティックの番号を入れます。 ◇リターンパラメータ　：Aレジスタにジョイスティックが向いている方向が入ります。 ◇使用レジスタ　　　　：すべて
00D8	GTTRIG	現在のトリガーボタンの状態を調べます。 ◇エントリーパラメータ：Aレジスタにトリガーボタンの番号を入れます。 ◇リターンパラメータ　：Aレジスタ＝0ならトリガーボタンは押されていません。 Aレジスタ＝255ならトリガーボタンが押されています。 ◇使用レジスタ　　　　：AF
0132	CHGCAP	CAPランプの状態を変えます。 ◇エントリーパラメータ：Aレジスタ＝0ならランプをつけ，≠0ならランプを消す。 ◇リターンパラメータ　：なし ◇使用レジスタ　　　　：AF
0020	DCOMPR (RST4)	HLレジスタとDEレジスタの内容を比較します。 ◇エントリーパラメータ：HL・DEの各レジスタに比較する値を入れます。 ◇リターンパラメータ　：HL＝DEならZフラグがたちます。 HL<DEならキャリーフラグがたちます。 ◇使用レジスタ　　　　：AF

さて，画面に何か文字を表示したいときはBIOSの00A2H番地を使いましょう。**リスト**1-3を見てください。これは現在のカーソル位置に`@′(アットマーク)を表示してBASICのコマンド待ちに戻るプログラムです。20行の`@′という書式は理解できますね？ **表**1-1の00A2H番地(CHPUT)のところを見るとわかるように，Aレジスタに文字コードを入れてCALL 00A2Hとすればよいわけです。またこのプログラムはD400H番地からメモリに入るので，アセンブルする前にBASICで

CLEAR100，&HD300↵

とするのを忘れないでください。でないとモニタのRコマンドを実行したときにエラーになります。

実行してみましたか？　GD400と押してリターン・キーを押すとすぐ

＊ＧＤ４００＠

となりましたね。リストの20行のアットマークを別の文字にすれば当然その文字が出ます。

こんどは好きな場所にアットマークを表示させてみます。カーソルの位置はRAM上の「ワークエリア」と呼ばれるところにいつも記録されています。具体的には，カーソルのX座標(1～40)がF3DDH番地，Y座標(1～32)がF3DCH番地です。ためしに，BASICのコマンド待ち(OKと出ている)状態でダイレクトにPRINT　PEEK(&HF3DD)としてみてください。そのときのカーソルの位置が何桁目かが表示されますね。

リスト1-3

```
  MSX Self Assembler  Rev 1.0     PAGE     1


 100:   D400                      ORG      0D400H
 110:
 120:   00A2 =         CHPUT      EQU      00A2H
 130:
 140:   D400 3E40                 LD       A,'@'
 150:   D402 CDA200               CALL     CHPUT
 160:   D405 C9                   RET
```

リスト1-4を見てください。160行でAレジスタに10を入れ，次にF3DCH，F3DDH番地にAレジスタ(すなわち10)を入れています。こうすればカーソルの位置を強制的に10行めの10桁めに設定できます。そこでさきほどの**リスト1-3**と同じことをすればそこ(10行めの10桁め)に＠が表示されるわけです。さっそく**リスト1-4**を入力，実行してみてください。実行の前に画面をクリアしておくと，＠が出るときはっきり見えますよ。

さて，次はこの＠を動かしてみます。キーボードからの入力によって動かすのがわかりやすくていいですね。キーボードからキーを入力するために，BIOSをもう１種類使いましょう。もう１度**表1-1**を見てください。009FH番地のCHGETが使えそうですね。キーから入力があるまで待つそうですから，リアルタイム・ゲームには応用できないでしょうが，こうした実習には最適です。**リスト1-5**をみてください。

009FHはキー入力があるとAレジスタに文字が入って戻ってくるのですが，今回はどのキーが押されたかに関係なく@を右に2桁動かすだけなので，戻ってきたAレジスタにそのまま現在のカーソルのX座標を入れ，+1して戻しています。入力された文字コードを使うならこんなことをしてはいけませんよ。

@を表示したあとはモニタに戻り，コマンド待ちになります。EC00H番地はモニタのスタート番地です。つづけて何度もGD400とすれば連続的に動くように見えます。

リスト1-4

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:   D400                      ORG      0D400H
110:
120:   00A2 =          CHPUT     EQU      00A2H
130:   F3DC =          CSRY      EQU      0F3DCH
140:   F3DD =          CSRX      EQU      0F3DDH
150:
160:   D400 3E0A                 LD       A,10
170:   D402 32DDF3               LD       (CSRX),A
180:   D405 32DCF3               LD       (CSRY),A
190:   D408 3E40                 LD       A,'@'
200:   D40A CDA200               CALL     CHPUT
210:   D40D C9                   RET
```

リスト1-5

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:   D400                      ORG      0D400H
110:                   ;
120:   009F =          CHGET     EQU      009FH
130:   00A2 =          CHPUT     EQU      00A2H
140:   EC00 =          MON       EQU      0EC00H
150:   F3DC =          CSRY      EQU      0F3DCH
160:   F3DD =          CSRX      EQU      0F3DDH
170:                   ;
180:   D400 3E10                 LD       A,10H
190:   D402 32DDF3               LD       (CSRX),A
200:   D405 32DCF3               LD       (CSRY),A
210:   D408 3E40                 LD       A,'@'
220:   D40A CDA200               CALL     CHPUT
230:   D40D CD9F00               CALL     CHGET
```

```
240:    D410 3ADDF3                 LD      A,(CSRX)
250:    D413 3C                     INC     A
260:    D414 32DDF3                 LD      (CSRX),A
270:    D417 3E40                   LD      A,'@'
280:    D419 CDA200                 CALL    CHPUT
290:    D41C C300EC                 JP      MON
```

3.暴走!?

ここでわざわざ「暴走」について説明しなくても，これまでの5本のリストを入力してすでに暴走させてしまったかたもいることでしょう。暴走はコンピュータが人間の犯した誤りに気づかず，誤りの通りにプログラムを実行してしまう動作を指すことばです。その動作は予想することが難しいのですが，以下のようになるのが「一般的な暴走」です。

①カーソルが2度と出てこない。当然キー入力はできない。

②リセットされてしまう。つまり電源ONと同じ画面になる。

③画面に関係ない文字が出る。ひどいときは画面いっぱいになるまで。

他にも何が起こるかわからないのが暴走です。暴走するとほとんどの場合キー入力ができなくなるので，せっかく入力したRAM上のプログラムが一切消滅してしまうことを覚悟の上で電源を切るしかありません。運よくキー入力ができるようになっても，暴走中にメモリにどんな「悪さ」をしたかもしれません。アセンブラ自体もRAM上にありますから，もしかすると悪影響を受けているかもしれないのです。いったん暴走したら，アセンブラからロードしなおすのが結局早道でしょう。

変な話ですが，1度あなたのMSXを暴走させてみませんか。さっき使った**リスト1-3**を改造して暴走させましょう。いちばん暴走させやすいのは，BIOSのエントリーアドレスをまちがえたときです。この例だと00A2H番地を使ってますね。これを002AH番地にまちがえてみましょう(**リスト1-6**)。暴走するとわかっていて実行するというのはどんな気分ですか。

リスト1-6

```
MSX Self Assembler  Rev 1.0     PAGE    1

100:    D400                    ORG     0D400H
110:
120:    D400 3E40               LD      A,'@'
130:    D402 CD2A00             CALL    002AH
140:    D405 C9                 RET
```

実行してみましょう。GD400のあとリターン・キーを押したが最後，2度とカーソルが出てきませんね。ご愁傷さま。暴走しました。CALL 002AHで2AH番地をCALLすると最終的には0030H番地から0205H番地に行ってしまい，「スロット」というものを見に行く「ややこしい」ルーチンを実行するのです。

このように暴走してしまってはどうしようもありません。キーをいくらたたいても戻ってきません。泣く泣く電源を切って，アセンブラを再ロードしましょう。ある程度の長さのプログラムを開発中なら，今のようにソースをテープにセーブもしてない状態で暴走してしまったら，もう泣くにも泣けません。こまめにテープにセーブしておくクセをつけましょう。リセット・ボタンがついているMSXでは電源を切るかわりにそれを押せばよいのですが，メモリはクリアされていると考えるのが順当でしょう。

いまの例のような典型的な暴走(予想外のルーチンを実行した)のほか，よくするまちがいが「無限ループ」です。ためしにBASICで

```
10 GOTO 10
```

としてRunさせてみてください。まるでマシン語が暴走したみたいに，カーソルが出ませんね。BASICの場合は CTRL + STOP でBreakされてOKになりますが，マシン語には(意図的にそのようなプログラムをつけ加えないかぎり)そんな便利な機能はありません。まだ「恐いもの見たさ」の気持ちが残っている人は，**リスト1-7**を実行してみてください。これは**リスト1-3**の140行にNEXTという名前(ラベル)をつけて，160行，すなわちBASICのコマンド待ちに戻るべきところでNEXTにジャンプさせたものです。こうすると@を表示しつづける恐怖の「無限ループ」となるのです。

もちろんこのループに入ってしまうと，もう飛び出す手段がありません。暴走と同じく，電源OFFかリセットだけです。いわゆる「暴走した」といわれている状態でも，この無限ループに入ってしまっていることが多いと思われます。

リスト1-7

```
  MSX Self Assembler  Rev 1.0     PAGE     1


 100:    D400                     ORG      0D400H
 110:
 120:    00A2 =       CHPUT       EQU      00A2H
 130:
 140:    D400 3E40    NEXT:       LD       A,´@´
 150:    D402 CDA200              CALL     CHPUT
 160:    D405 C300D4              JP       NEXT
```

4. ワークエリアを使う

次は，ちょっと高級に「ワーク・エリア」というものを使ってみましょう。

ワーク・エリアは，日本語にすると「作業領域」です。もう少しくだけた言いかたにすると「仕事場」ですね。仕事場というだけあって,用途はさまざまです。主に次のように使います。

①値が変化しそうな「意味のある数」をメモしておく。

②そのまま置いておくと壊されそうなデータを逃がしておく。

③あるルーチンやエリアのうち，将来変更されそうなものの先頭番地を記録しておく。

④フック。フックの意味と使いかたは第5章の2.で説明します。

先に画面に文字を出したとき，カーソルの位置がF3DCH，F3DDH番地に入っていると書きました。これもワーク・エリアです。使いかたとしては①に当たります。カーソルの位置などはそれこそ刻々と変化するものですから，RAM上にないとまずいわけですね。**表**1-2にMSXのワーク・エリアの一部をあげておきます。意味を見て使い道を想像してみてください。重要なMSXのワーク・エリアは，使うごとに説明してゆきます。

さて，実際にワーク・エリアを使ってみることにしましょう。①の例はカーソルの位置で体験しましたから，②を実習することにします。

リスト1-8を見てください。これはメモリのD480H番地とD481H番地の内容を交換するプログラムです。Aレジスタ以外のレジスタを使えばこんなややこしいことをしなくてもいいのですが，ここはサンプルのため少々長くなっています。

190行の“DEFS”命令(擬似命令)の意味・使いかたは知っていますね。これは“WORK”という名で1バイト分のワーク・エリアを確保し

ておくようにアセンブラに指示する命令です。こうすることによって，D413H番地は1バイトのワーク・エリアとしての役割が与えられるのです。

プログラムはLD命令の行進となってしまいました。WORKにD480H番地の内容を退避させておいてD481H番地→D480H番地の内容の移動をし，WORKの内容をD481H番地に戻しています。

表1-2　主なワーク・エリア

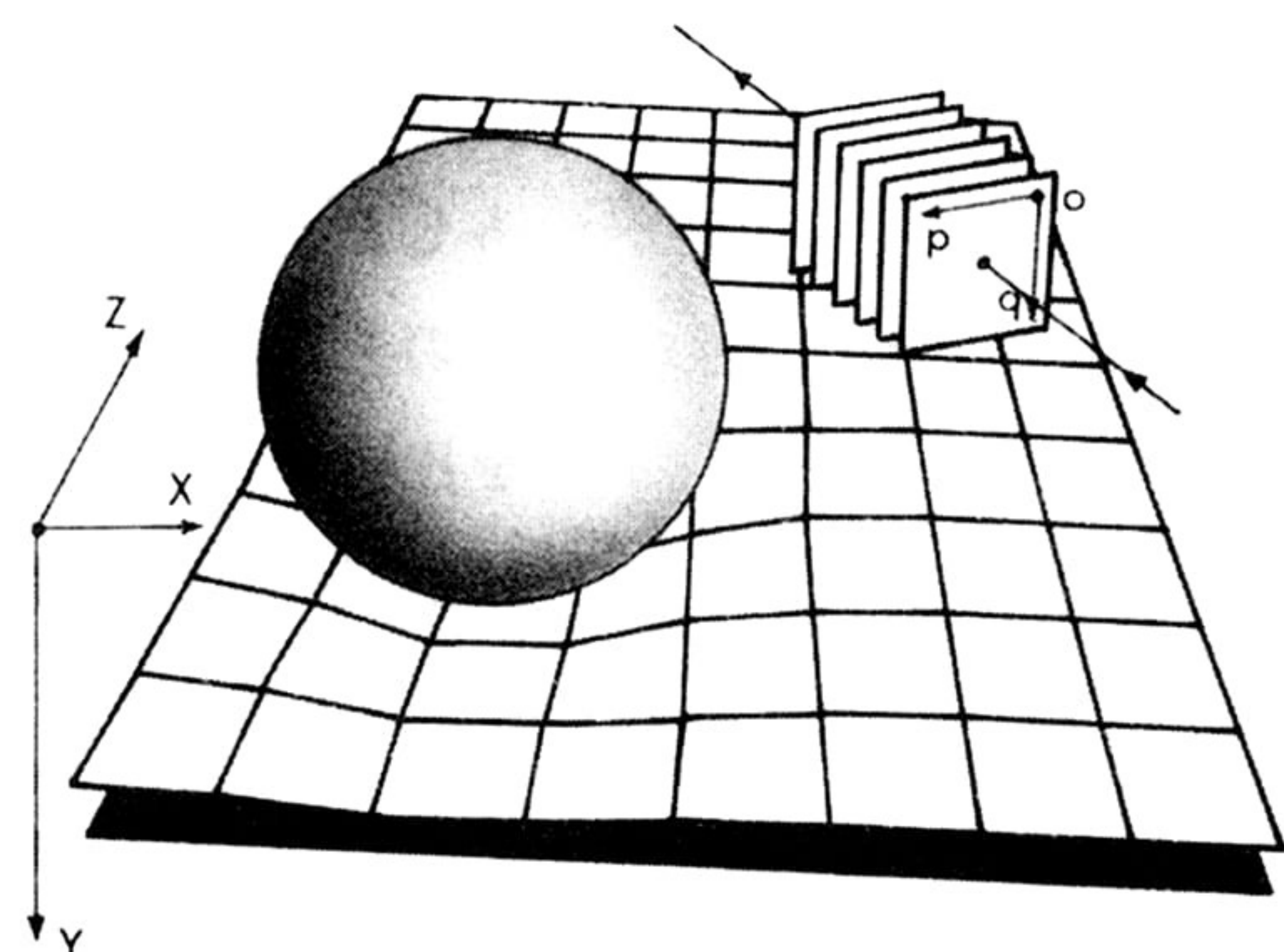
番地	ラベル	長さ	意味
F39A	USRTAB	20	USR関数の機械語プログラム（0～9）の開始番地。機械語プログラム定義前の値はすべてFCERR（エラールーチン）
F3DC	CSRY	1	カーソルのY座標
F3DD	CSRX	1	カーソルのX座標
F55E	BUF	n	nはBUFLEN＋3。タイプインした文字が入るバッファ
F672	MEMSIZ	2	メモリの最上位番地
F674	STKTOP	2	スタックとして使える最上位番地。CLEARで変えられる
F676	TXTTAB	2	テキストエリアの先頭番地。INITで初期設定した後は変化しない
F678	TEMPPT	2	テンポラリーディスクリプタ（TEMPST）の空きエリアの先頭番地。初期設定ではTEMPSTを指す
F67A	TEMPST	n	nは3×NUMTMP。
F69B	FRETOP	2	文字フリーエリアの先頭番地
F6C2	VARTAB	2	配列でない変数エリアの開始番地。プログラムの大きさが変わった場合にはこの値は変化する。NEWでプログラムを消すと，〔TXTTAB〕＋2にセットされる
F6C4	ARYTAB	2	配列テーブルの開始番地。配列でない，新しい変数を見つけると，6ずつ増やし，CLEARCが〔VARTAB〕に値をセットする
F6C6	STREND	2	使用中のメモリーの最後の番地。変数を追加すると，この値は増加し，CLEARCが〔VARTAB〕に値をセットする
F7F6	DAC	16	十進数演算のワークエリア
F85F	MAXFIL	1	ファイル番号の最大値
FC4A	HIMEM	2	利用可能なメモリーの最上位番地

フック

番地	ラベル	意味　呼び出しモジュールと使用目的
FD9F	H.TIMI	MSXIO　タイマー割り込み処理。タイマーで起動する割り込みを追加するため
FDA4	H.CHPU	MSXIO　CHPUT（1文字出力）の始め。他のコンソール出力装置をつなぐため
FDDB	H.PINL	MSXINL　PINLIN（プログラム行入力）の始め

FECB	H. RUNC	BIMISC RUNC (RUNのためのクリアー)
FEFD	H. ERRP	BINTRP ERRPRT (エラー表示)
FF43	H. GONE	BINTRP
FF89	H. LIST	BINTRP LIST

リスト1-8

```
MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                    ORG      0D400H
110:    D400 3A80D4             LD       A,(0D480H)
120:    D403 3213D4             LD       (WORK),A
130:    D406 3A81D4             LD       A,(0D481H)
140:    D409 3280D4             LD       (0D480H),A
150:    D40C 3A13D4             LD       A,(WORK)
160:    D40F 3281D4             LD       (0D481H),A
170:    D412 C9                 RET
180:
190:    D413          WORK:     DEFS     1
```

これだけは知っておこう

2 ちょっと高級なことも覚えておこう

1. フラグ条件分岐・比較命令の使いかた

これからはLD以外の命令を使って重要な処理を扱っていきます。まず，場合によって処理の手順を変更する方法の基本から。

単に数字を足したり引いたりするだけのプログラムならともかく，ふつうは「もし〜ならこっち，でなければあっち」というかたちの処理をしなければなりません。BASICならIF文で，はっきり目に見えるように場合分けができます。たとえばAという変数の値が5よりも大きいか小さいか，それともちょうど5なのかということを知りたいとき，BASICでは**リスト1-9**のようにできます。IFのあとの数式のところに比較したい2数を記号で結んで書けばよいのですから，非常にわかりやすいですね。こんなところが「高級言語」と呼ばれる理由になっているのです。

リスト1-9

```
          ⋮
200     IF    A>5    THEN   (大きいときの処理)
210     IF    A<5    THEN   (小さいときの処理)
220    (等しいときの処理)
```

ところがさして高級でもないマシン語で数値の比較を表現すると，ちょっと見ただけではわからなくなってしまいます。Z80の比較命令はCPというニモニックで表わされ，CPのあとのオペランドにレジスタ名や数を書いて，常にAレジスタとそれとを比較します。直接BレジスタとCレジスタとを比べることはできないので注意してください。もしそうしたければ，Aレジスタの中身が壊されないような工夫をしたのちBレジスタの内容をAレジスタに移し，そして初めてAとCとを比較する，という手順を踏まねばなりません。このあたりの**リスト**は1-10です。比較するものの一方はいつもAレジスタでなければいけないということを忘れないでください。また，比較したあともAレジスタの内容は比較前と変わりません。

リスト1-10

```
10    (Aレジスタの内容を退避)
20    LD    A, B
30    CP    C
```

さて，比較した結果はどうやって知るのでしょうか。ここでフラグが活躍します。フラグは，何か計算をした結果の状態を示す「旗」です。言ってみれば見張り番のようなもので，「今した引き算の答はゼロだ」とか「足し算をしたらレジスタの内容があふれた」といった状態を旗の上げ下げで示しているのです。Z80のフラグは6種類あり，「フラグ・レジスタ(Fレジスタ)」としてまとめられています。フラグについて詳しいことは「マシン語入門・基礎編」に書かれているので，そちらを参照してください。

では，実習に入りましょう。例として先にBASICで書いた**リスト1-9**に似た処理をマシン語で書くことにします。**リスト1-11**を見てください。これは「1文字入力されるまで待ち，もし入力された文字が“A”なら“O(オー)”をプリント(マルの代わり)，“A”以外なら“X(エックス)”(バツの代わり)をプリントする」というプログラムです。ミソは150～170行にあります。150行のBIOSコールでAレジスタに1文字入力し，それを160行のCP命令で16進数の41Hと比較しています。41Hは英大文字のAのキャラクタ・コードですね。知らなかった人は自分のMSXのマニュアル巻末にある「キャラクタ・コード表」を見てください。

フラグが活躍するのは170行です。これはジャンプ命令の一種で，条件ジャンプ命令と呼ばれます。条件ジャンプ命令は各フラグが条件に合えばジャンプし，合わなければジャンプしません。170行ではZ(ゼロ)

リスト1-11

```
 MSX Self Assembler   Rev 1.0      PAGE      1


100:    D400                       ORG      0D400H
110:
120:    009F =          CHGET      EQU      009FH
130:    00A2 =          CHPUT      EQU      00A2H
140:
150:    D400 CD9F00                CALL     CHGET
160:    D403 FE41                  CP       'A'
170:    D405 2806                  JR       Z,ONAJI
180:    D407 3E58                  LD       A,'X'
190:    D409 CDA200                CALL     CHPUT
200:    D40C C9                    RET
210:
220:    D40D 3E4F       ONAJI:     LD       A,'O'
230:    D40F CDA200                CALL     CHPUT
240:    D412 C9                    RET
```

フラグが立っていれば"ONAJI"というラベルがついている行にジャンプします。Zフラグが立つということはその前の命令(CP命令)で結果がゼロになったということです。つまりA＝41Hだったということですね。

もし150行で入力した文字が"A"でなかったらZフラグが立ちませんからジャンプせずに180行に進み，エックスをプリントします。もし"A"だったらZフラグが立って"ONAJI"の220行にジャンプし，オーをプリントします。

以上はある数が他の数と等しいかどうかを調べる例でした。等しくないとき，大小を調べるのにはどうするのでしょう。それにはC（キャリー)フラグを使います。Cフラグは，比較の対象がAレジスタの内容よりも大きかったときに立ちます("CP□□"でA<□□のとき)。Aレジスタの値と同じかまたは小さかったときにはフラグをおろします。

リスト1-12を見てください。これは，入力されたキーのキャラクタ・コードが16進数の40未満かどうかを調べるものです。「未満」ですから，40Hが入ってはいけません。3FH以下でなくてはならないのです。150行で1文字入力し，160行で40Hと比較しています。もし入力した文字のコードが40Hかそれ以上ならCフラグが立たず，170行のジャンプ命令は実行しないで180行に進みます。もし入力した文字のコードが3FH以下ならCフラグが立つので"SHOU"というラベルのついた220行に

リスト1-12

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                      ORG      0D400H
110:
120:    009F =          CHGET     EQU      009FH
130:    00A2 =          CHPUT     EQU      00A2H
140:
150:    D400 CD9F00               CALL     CHGET
160:    D403 FE40                 CP       'A'-1
170:    D405 3806                 JR       C,SHOU
180:    D407 3E58                 LD       A,'X'
190:    D409 CDA200               CALL     CHPUT
200:    D40C C9                   RET
210:
220:    D40D 3E4F       SHOU:     LD       A,'O'
230:    D40F CDA200               CALL     CHPUT
240:    D412 C9                   RET
```

ジャンプし，マルをプリントします。

ここで注意しなければいけないのは，キャリーが立った(Aレジスタの内容がオペランドよりも小さい　(A)＜op)からといって，その逆，すなわち「キャリーが立たなかった（A)＞op」は成り立たないということです。おわかりですか？ キャリーが立たないのは「(A)＜opでない」ということなので「(A)＞op」ではなく「(A)≧op」なのです。CP命令のあと「JP　NC,～」でジャンプさせようとするときは(A)＝opの場合が含まれていることを忘れないように。

つまり，CP命令のオペランドに置いた数とAレジスタの内容の値との関係は**表1-3**のようになるわけです。キャリーフラグが立つかどうかを判断する感覚は慣れで養えます。慣れはすなわち実践です。このリストを改造してどんどん暴走させてください。

表1-3

○(A) はAレジスタの内容を表わす
○OP はCP命令のオペランドを表わす

① (A)＝OP　のとき　Zフラグが立つ
　　　　　　　　　JP　Z,～で JP

② (A)＜OP　のとき　Cフラグが立つ
　　　　　　　　　JP　C,～で JP

条件判断の第2段階に進みましょう。**表1-3**にない条件を判断してみるのです。まず(A)＞opの判断から。**表1-3**①②のどちらも成り立たなかったとき，すなわちNZかつNCのとき(A)＞opと考えられます。

(A)＜＞opは？ もちろんNZですね。(A)≦opは？ この場合はZとCの2つのフィルターに通さねばなりませんね。ここらへんをop＝5として**表1-4**にまとめておきました。よく見て納得してください。

これまではZフラグとCフラグの2つについて解説しました。Z80

表1-4

①A＝5の判断
　CP　5
　JP　Z, [A＝5]
[A≠5]

②A＜＞5の判断
　CP　5
　JP NZ, [A≠5]
[A＝5]

③A ＜5の判断
　CP　5
　JP　C, [A＜5]
[A≧5]

④A＞5，A≦5の判断
　CP　5
　JP　Z, SONOTA
　JP　C, SONOTA
[A＞5]
SONOTA：[A≦5]

⑤A≧5の判断
　CP　5
　JP　NC, [A≧5]
[A＜5]

にはこのほかに4つのフラグがあります。S，H，P/V，Nです。HとNは条件判断として使うことができません。これらはBCD演算をするとき，CPU自身が内部で参照するためにだけ使われます。また，Sは符号ビットで，AレジスタのMSB（第7ビット）がそのまま入ります。符号つき2進数を扱うときに使われます。P/Vフラグは直前の演算が論理演算だったか算術演算だったかによって2種類の異なる意味を持ちます。論理演算だったときはパリティフラグとして機能し，Aレジスタの8ビットのうち，1になっているビットがいくつあるかを示しています。もし1のビットが偶数個であればP/Vフラグがセットされニモニックの条件判断はPE，1のビットが奇数個ならP/Vフラグがリセットされニモニックの条件はPOとなります。

算術演算だったときはオーバーフローフラグとして機能します。これは直前の算術演算の結果（Aレジスタの内容）が2つの補数の範囲(10進数で＋127～－128)を超えると(すなわちオーバーフローすると)セットされます。ニモニックの条件表現はオーバーフローのとき（フラグが立ったとき)がPE，そうでないときがPOです。2の補数やBCDなどの数値表現については，マシン語入門・基礎編および類書で勉強してください。

2.シフト，ローテイト命令の使いかた

さあ，次はシフト・ローテイト命令を取り上げます。動作自体は簡単でわかりやすいはずなのに，種類が多いためにZ80に相当慣れ親しんだ人でも命令表と首っぴきでなくては恐くて使えないというしろものです。意外に多く使いみちがあるので，これらの命令の「考えかた」だけは知っておきましょう。 実際に使うときは命令表，動作図を見ながらで十分なのですから。

シフトもローテイトも，あるレジスタの内容全体をビット単位で動かす命令です。動かす方向やビット数には各2通りあって，場合によって選べます。

「シフト」は「移動」と訳され，あるレジスタの内容全体を1ビット単位で左または右方向に桁移動させる命令です。たとえば，図1-3のようにビット0をビット1に，ビット1をビット2に……と順繰りに移動してゆけば，「左シフト」が完成します。

ここでビット7の内容はどこに移るのでしょう？ ビット7の行き場は決まっています。それはCフラグです。右シフトでも同じで，ビット0はCフラグに入ります。また，図1-3におけるビット0（右シ

フトならビット7）には新たに何かが入らねばなりませんが，それは何でしょう。これには2通りあります。すなわち0か以前のビットのままかのいずれかです。シフト前と同じ値が入るということは，たとえば図1-3ならシフト前のビット0の値はシフト後のビット1とビット0の両方に入るということです。

次にローテイトです。「ローテイト」は「回転」と訳され，あるレジスタの内容(あるいはそれプラスCフラグ)を環状につなげて1ビット

図1-3

7 6 5 4 3 2 1 0 ビット番号

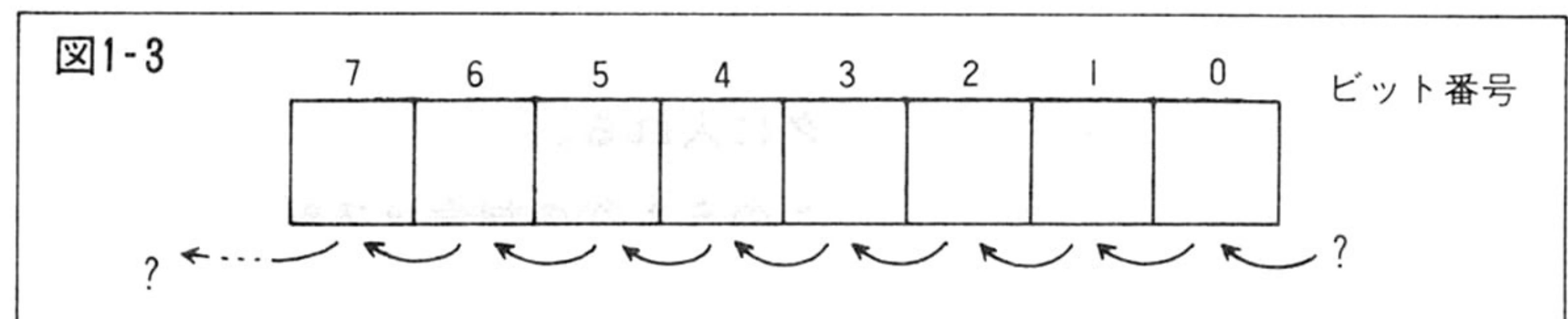
? ?

または4ビット単位で左右に回転移動させる命令です。ローテイト命令はシフト命令よりもバラエティに富んでおり，回転の単位が1ビットだけでなく4ビットにもなるところが特徴です。Z80のシフト命令のニモニックと動作図を図1-4に，ローテイト命令を図1-5にまとめておきました。折にふれて見てください。

図1-4

CY b_7 b_0 0 SLA (Shift Left Arithmetic)

CY b_7 b_0 SRA (Shift Right Arithmetic)

CY b_7 b_0 0 SRL (Shift Right Logical)

図1-5

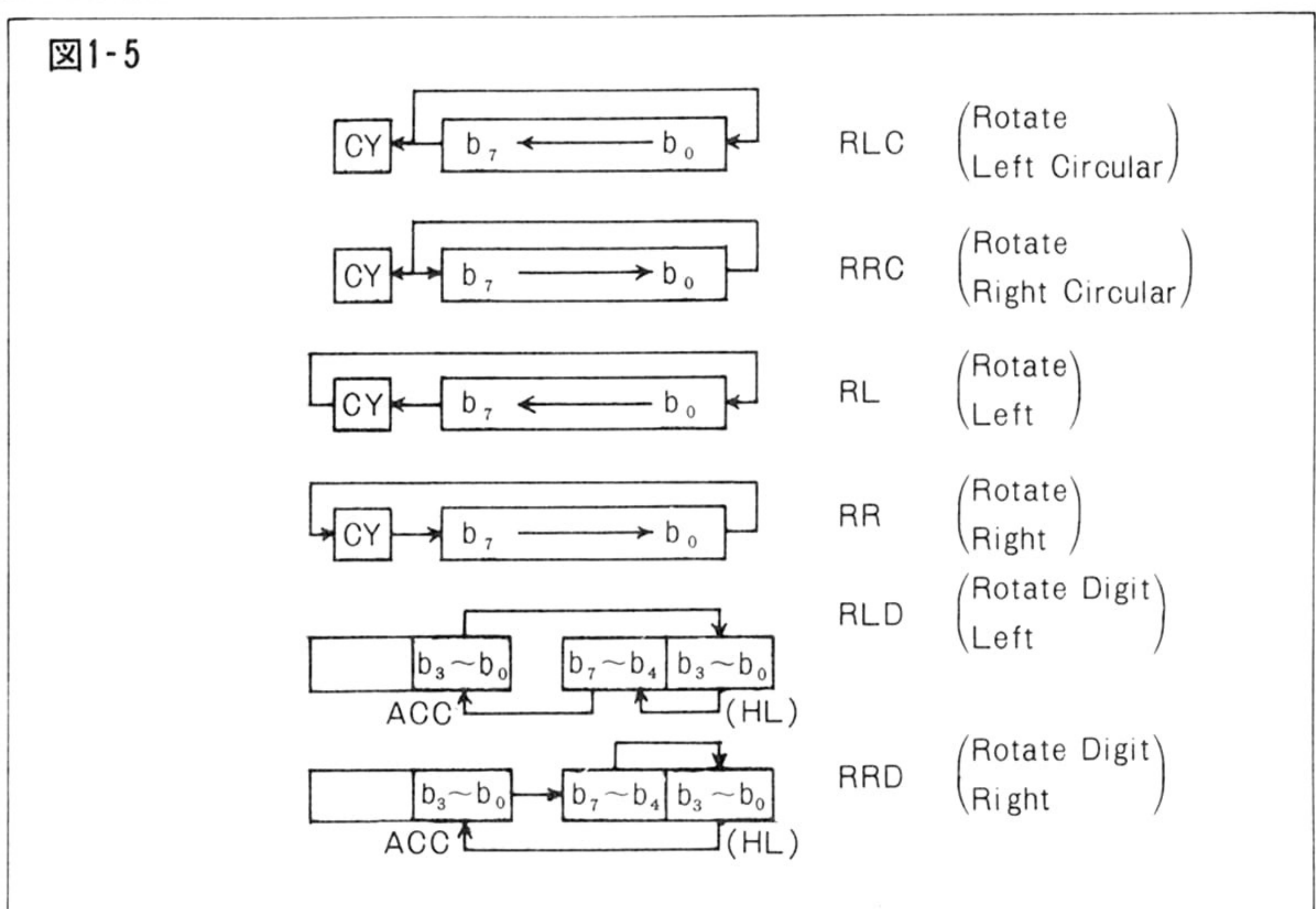
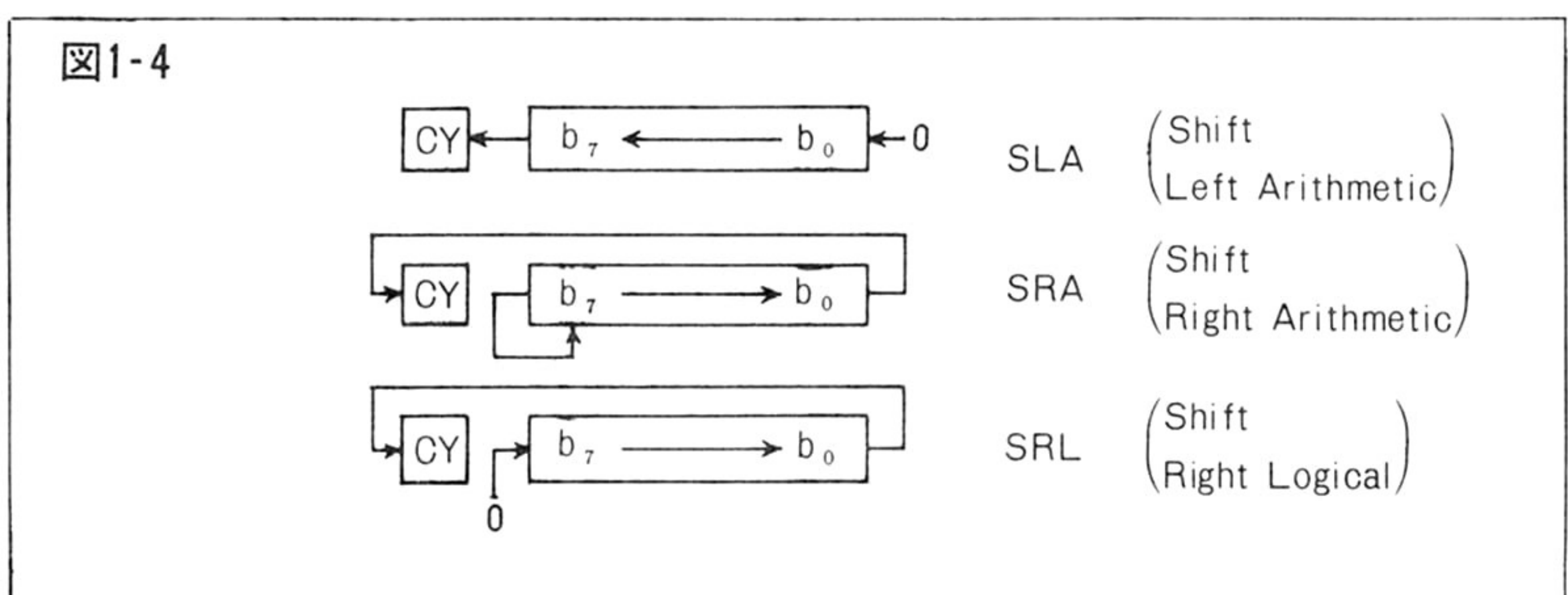
CY b_7 b_0 RLC (Rotate Left Circular)

CY b_7 b_0 RRC (Rotate Right Circular)

CY b_7 b_0 RL (Rotate Left)

CY b_7 b_0 RR (Rotate Right)

$b_3 \sim b_0$ ACC $b_7 \sim b_4$ $b_3 \sim b_0$ (HL) RLD (Rotate Digit Left)

$b_3 \sim b_0$ ACC $b_7 \sim b_4$ $b_3 \sim b_0$ (HL) RRD (Rotate Digit Right)

これらの「ややこしい」命令は，いったいどんなときに使われるのでしょう。シフト命令から見てゆきます。シフト命令は，主に以下のような目的で使います。

①レジスタの内容が数値を表現しているとみなし，2のべき乗倍，あるいは2のべき乗分の1する。

②レジスタの内容の特定ビット(または隣接するビット群)をそのレジスタの端に運ぶ。

③レジスタの内容の特定1ビットが1か0かを調べるために，Cフラグに入れる。

このうち③の判定はZ80のBIT命令でもできるのですが，ループの中で使うときはシフト命令のほうが便利なときもあります。また②はのちに論理演算を解説するときにあわせて説明しましょう。ですからここでは，①について取り上げます。

さて，いったい「2進数を2倍する」とはどういうことでしょう？**図1-6**を見てください。これは2進数の01010100(10進で84)を1ビットずつ左シフトしたところです(SLA)。実行前のCフラグの値は不定です。実行後の値は10進数で168になっていますね。これは実行前のちょうど2倍です。つまり，左シフトするということはそのレジスタの内容を2倍するということなのです。もう1回シフトしてみましょうか。**図1-7**は，**図1-6**の実行後の168をもう1ビット左シフトした図です。こんどはCフラグが立ってレジスタの内容が10進で80になりましたね。Cフラグを第8ビットとして見，Cフラグ＝1なら256ということにしておけば256＋80＝336となり，168の2倍です。こうして，積が511までの数なら1回の左シフトで計算できるのです。もちろん，何回かつづけてシフトすれば2のべき乗倍(2^n，ただしnはシフトの回数)のかけ算ができるわけです。

図1-6

	C									
実行前	?	0	1	0	1	0	1	0	0	84
実行後	0	1	0	1	0	1	0	0	0	168

図1-7

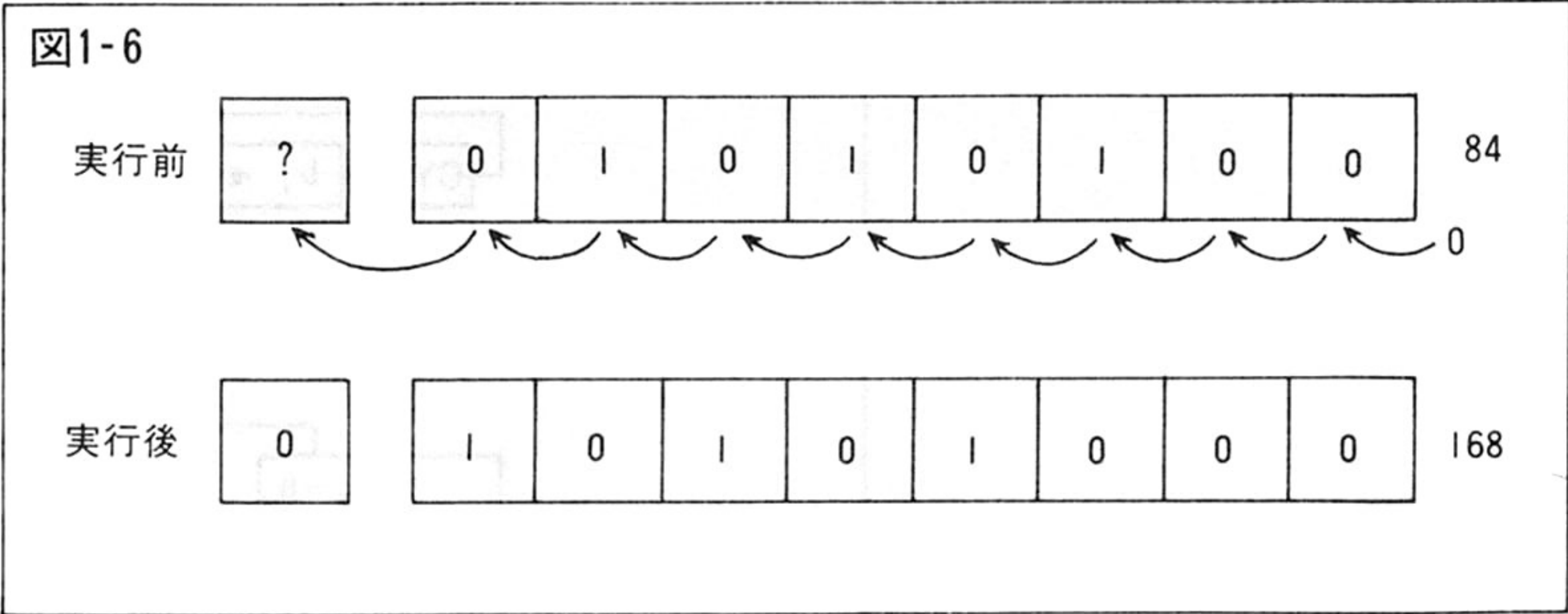
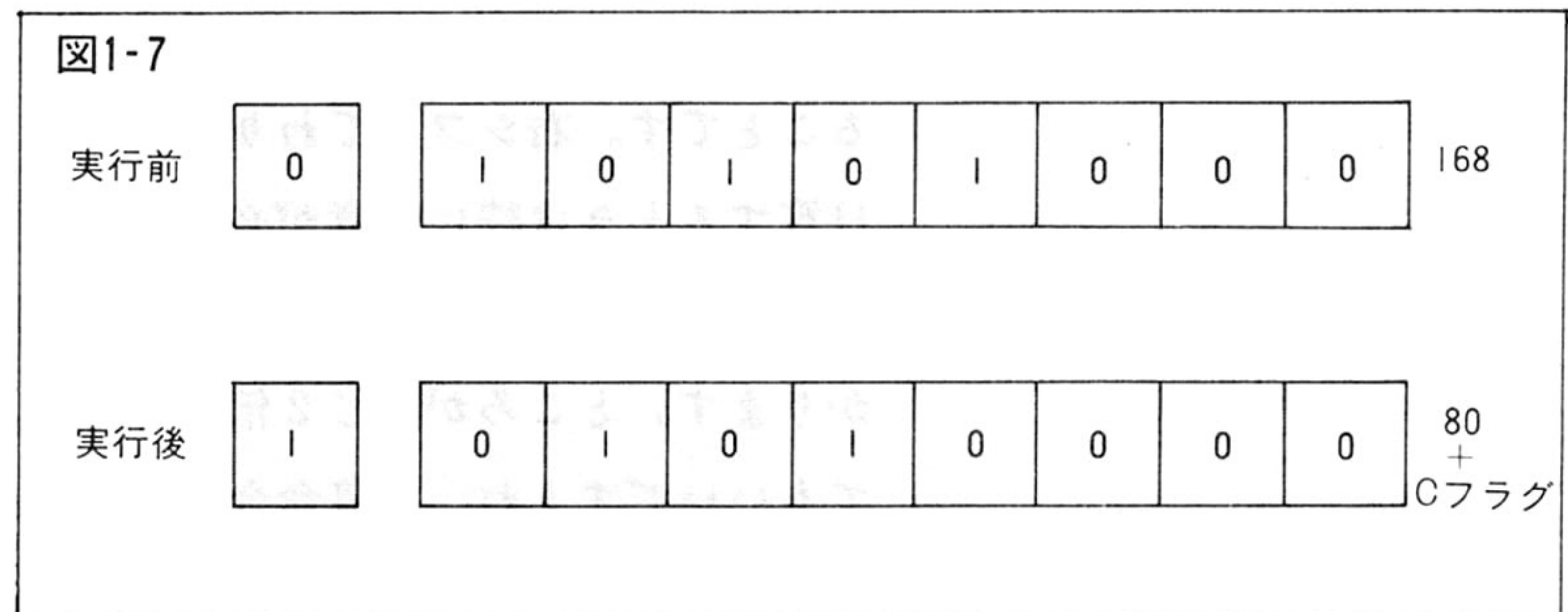
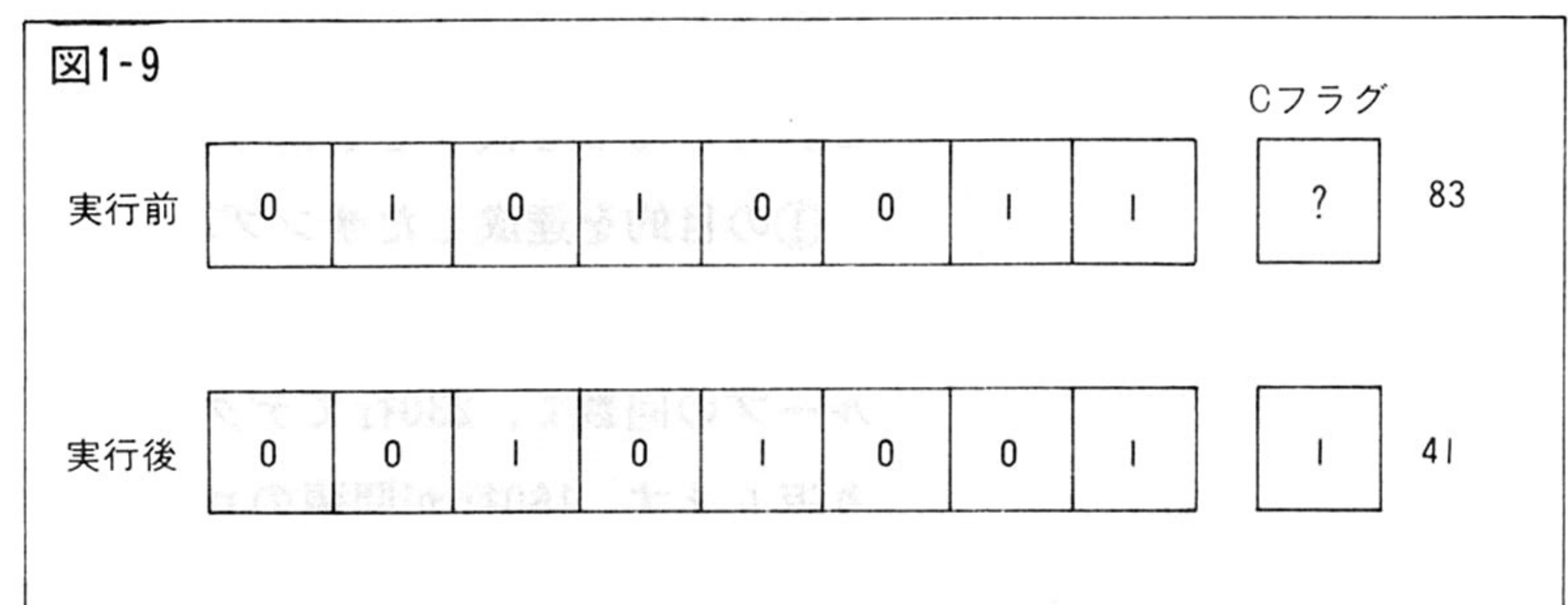
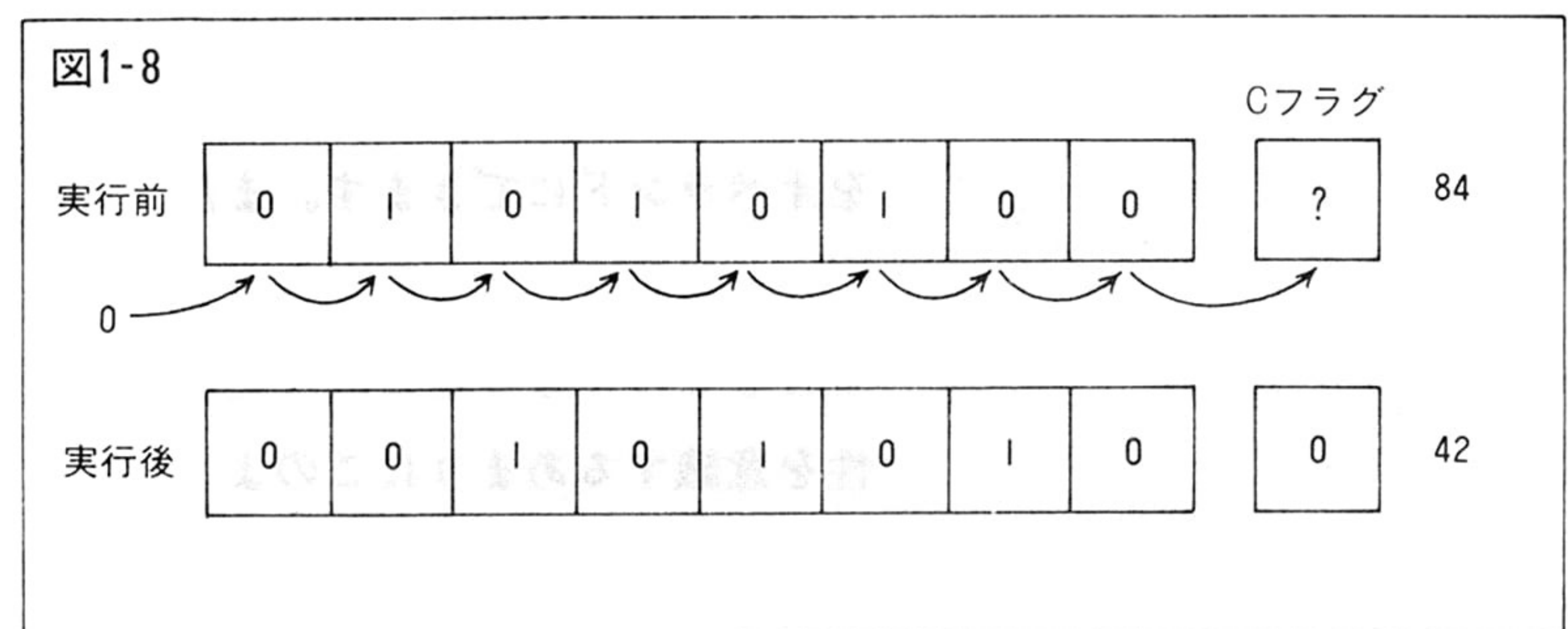
実行前	0	1	0	1	0	1	0	0	0	168
実行後	1	0	1	0	1	0	0	0	0	80 + Cフラグ

こんどは½です。カンのいい人ならもうおわかりでしょう。そう，½倍なら右シフトを１回すればいいのですね。図1-8を見てください。これも10進の84を右に１ビットした図です(SRL命令)。実行後は42と予想どおり½になりましたね。しかしこれはシフト前の数が偶数だったときの話。奇数だとちょっと話がちがってきます。図1-9を見てください。10進数で83を右シフトしてみました。実行後は41になってしまいましたね。そのかわりＣフラグが立っています。つまり，単純に½した結果の小数部分を切り捨てたものになっているわけです。Ｃフラグが立っているのは切り捨てが起きたときを示していると理解すればよいですね。これも１回のシフトなら½ですが，ｎ回シフトすれば$1/2^n$したのと同じことになります。ただし左シフト(乗算)のときとはちがって最初のレジスタの内容は８ビット分(255まで)でなくてはだめで，しかも奇数だと切り捨てが起こります。

図1-8

									Cフラグ	
実行前	0	1	0	1	0	1	0	0	?	84
実行後	0	0	1	0	1	0	1	0	0	42

0 →

図1-9

									Cフラグ	
実行前	0	1	0	1	0	0	1	1	?	83
実行後	0	0	1	0	1	0	0	1	1	41

ただ，ここまででひとつ注意しておきたいことは，実行時間に関することです。右シフトでわり算をするときはともかく，左シフトでかけ算するときは特に注意が必要です。たとえばAレジスタの内容を2倍したいときにSLA命令を使うとMSXでは10ステートの実行時間がかかります。ところが同じ2倍するなら，Aレジスタ＋Aレジスタとしてもいいですよね。加算命令のひとつADD A，Aを使えばMSXで5ステートと半分ですみます。単に2のべき乗倍するだけなら加算を並べたほうがずっと速いわけです。しかし対象レジスタがAレジスタでないときはいったんAレジスタにロードしなくてはなりません。8ビットレジスタどうしのロード命令はやはりMSXで5ステート（しかも加算・ロードとも1バイト命令）ですから，ロード＋加算とシフト1発とではステート数もオブジェクトのバイト数も変わりありません。ソースが長くなるだけです。

さて，次はローテイト命令です。ローテイト命令は主に次のようなときに使われます。

①対象レジスタのビット内容のうち，特定の部分を取り出す。取り出し口はCフラグから。

②4ビット単位でローテイトしてBCD演算のときに使う。

①はシフト命令でもできますが，ローテイト命令なら8回ローテイトすれば元に戻るところが大きな特徴です。①の目的で使われるローテイト命令はRLC，RL，RRC，RR命令です。これらの命令はみな任意の8ビット汎用レジスタとHL，IX，IYなどで示されたメモリの内容をオペランドにできます。また，これらの命令には特別にAレジスタだけを対象にしたものもあります。Aレジスタだけを対象にしたRLCAなどの4命令はZ80の先祖である8080時代の名残りで，上位互換性を意識するあまりにこのようなすっきりしない命令系統になってしまったのでしょう。Aレジスタ専用のRLCA命令は1バイトで5ステート(MSXの場合)，AレジスタをオペランドにしたRLC命令であるRLC A命令は2バイトで10ステート(同)なので，Aレジスタを対象にこれらの命令を使うときはちゅうちょせずRLCAを使いましょう。

①の目的を達成したサンプルが**リスト1-13**です。これはAレジスタの内容の8ビットが1か0かを画面に表示するものです。140行の8はループの回数で，230行でデクリメントしてゼロでなければ150行から繰り返します。160行が問題のローテイト命令で，今回は目的のレジスタがAなのでRLAを使いました。RLAは**図1-5**のような動作をする命令

で，Aレジスタの内容をCフラグを含めて左方向に回転し，ビット0には回転前のCフラグが入ります。これを1回実行すると，ビット7の内容がCフラグに入ります。170行でそれが1だったかどうか判断し，もし0だったら(すなわちCフラグが立っていない=NC) ZEROと名のついたところに飛びます。このとき，Dレジスタにはすでに`0′のキャラクタ・コードが入っているので，210行で`0′の文字がプリントされます。190，200行はレジスタの内容が壊されないように逃がしているのです。220行でそれを戻したあと，8ビット分終わったかどうか見て，まだなら次のビットを同じように試します。

もし170行でC=1(Cフラグが立っている)ならこの条件は成り立たず，ジャンプしないで180行に進みます。180行では新たにDレジスタに`1′のキャラクタ・コードを入れています。ここはINC　Dでもかまいません。また，160行はRLCAでも結構です。RLAとちがうところはCフラグを環の中に含むかどうかで，今回の場合ビット0～7が回転してくれさえすればよいのですから大差ありません。160行の命令をRLCAに変えて実行してみてください。RLC　AでもRL　Aでも同じです。

実行したとき改行されないのが見にくいという人は，BASICからUSR文で呼び出せば見やすくなります(DEFUSR=&HD400：A=US

リスト1-13

```
  MSX Self Assembler  Rev 1.0     PAGE     1


 100:    D400                     ORG      0D400H
 110:
 120:    00A2 =         CHPUT     EQU      00A2H
 130:
 140:    D400 0E08                LD       C,8
 150:    D402 1630      START:    LD       D,'0'
 160:    D404 17                  RLA
 170:    D405 3002                JR       NC,ZERO
 180:    D407 1631                LD       D,'1'
 190:    D409 47        ZERO:     LD       B,A
 200:    D40A 7A                  LD       A,D
 210:    D40B CDA200              CALL     CHPUT
 220:    D40E 78                  LD       A,B
 230:    D40F 0D                  DEC      C
 240:    D410 20F0                JR       NZ,START
 250:    D412 C9                  RET
```

R(0))。なお，実行結果が信じられない人は，モニタのXコマンドでAレジスタの内容を確認してから実行してみるとよいでしょう。

②については，BCD表現の演算を知っていることが必要です。BCD表現とその演算については別項で詳しく述べますので，そちらをごらんください。

最後に，複数バイトにわたるローテイトを取り上げてみましょう。RLC, RRC, RL, RR命令は1命令で8ビット＋Cフラグを左右に1ビット回転させますが，これらをシフト命令と組み合わせると多精度のローテイトができるようになるのです。

ミソはローテイトにCフラグが含まれていることです。図1-10を見てください。ここでは3バイト(24ビット)の2進数を左に1ビット回転させています。まず(1)で，3バイトのうち最上位バイトの最上位ビットをCフラグに入れておきます。これは必ず最上位バイトを壊さないように他のレジスタなどに移して実行します。(2)からは通常の左ローテイト(RL)です。3バイトのうち最下位バイトから順に実行します。(5)まで進むと3バイトはきれいに1ビットずつローテイトしていますね。つづけてもう1ビットローテイトするときは再び(1)から繰り返します。

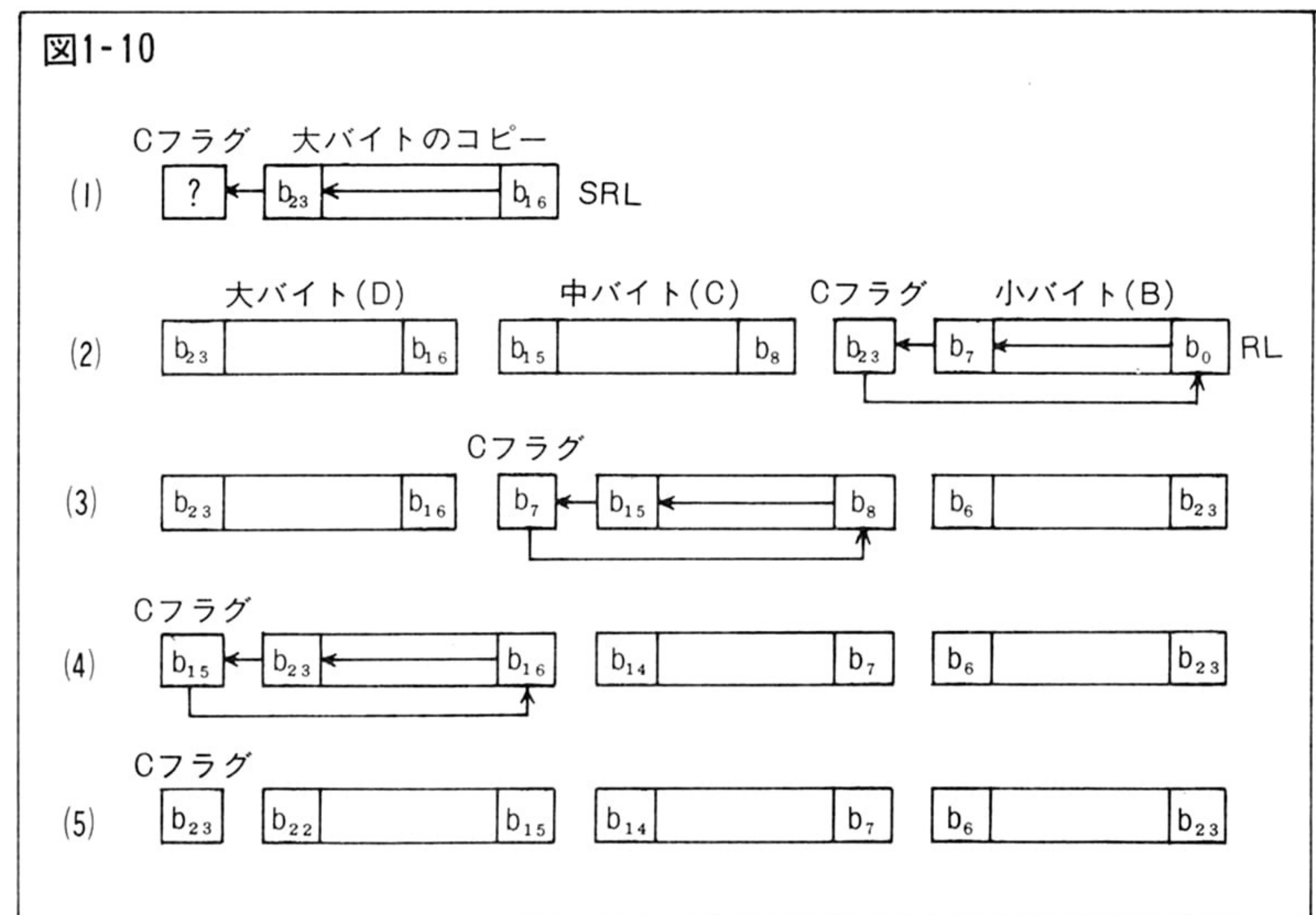
図1-10

(1) Cフラグ [?] ← 大バイトのコピー [b_{23} … b_{16}] SRL

(2) 大バイト(D) [b_{23} … b_{16}]　中バイト(C) [b_{15} … b_8]　Cフラグ [b_{23}] ← 小バイト(B) [b_7 … b_0] RL

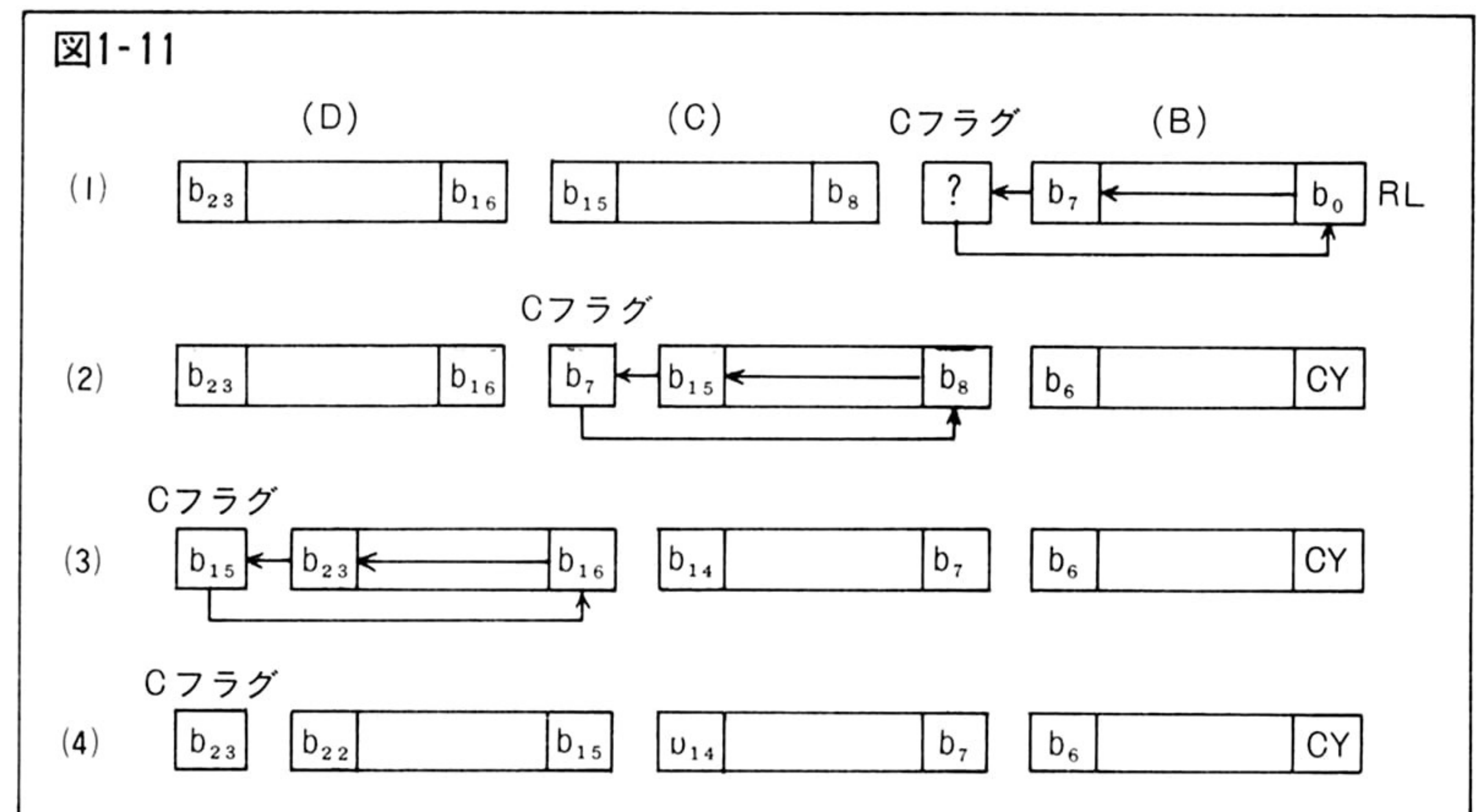
(3) [b_{23} … b_{16}]　Cフラグ [b_7] ← [b_{15} … b_8]　[b_6 … b_{23}]

(4) Cフラグ [b_{15}] ← [b_{23} … b_{16}]　[b_{14} … b_7]　[b_6 … b_{23}]

(5) Cフラグ [b_{23}]　[b_{22} … b_{15}]　[b_{14} … b_7]　[b_6 … b_{23}]

以上は1バイトにおけるRLC命令，すなわちCフラグを含めないローテイトの3バイト版でした。これをCフラグを含めたもの（1バイトにおけるRL命令)にしたければ図1-11のようにすればよいでしょう。これら2つの相違点は，図1-10での(1)を省略していることです。図1-11において実行前にCフラグをリセット(ゼロにする)しておけば，3

バイトのシフト(SLA)と同じことになります。他に一切影響を与えずにCフラグだけをリセットする方法は，のちに論理演算を取り上げるときに述べます。

図1-10を実際のプログラムにしたものが**リスト1-14**，**図1-11**は**リスト1-15**です。これらはどちらも3バイトのデータをB，C，Dレジスタに入れており，この順で最上位→最下位バイトとしています。110行～130行であらかじめこの3つのレジスタの初期値を44H（2進数で01000100）にしました。この数なら左に1ビットずれたとき10001000すなわち88Hとなるのでわかりやすいでしょう。あとは図のとおりにしています。アセンブルして実行後，モニタのXコマンドでレジスタの

図1-11

リスト1-14

```
 MSX Self Assembler  Rev 1.0     PAGE    1


100:    D400                   ORG     0D400H
110:    D400 0644              LD      B,44H
120:    D402 0E44              LD      C,44H
130:    D404 1644              LD      D,44H
140:    D406 7A                LD      A,D
150:    D407 CB3F              SRL     A
160:    D409 78                LD      A,B
170:    D40A 17                RLA
180:    D40B 47                LD      B,A
190:    D40C 79                LD      A,C
200:    D40D 17                RLA
210:    D40E 4F                LD      C,A
220:    D40F 7A                LD      A,D
230:    D410 17                RLA
240:    D411 57                LD      D,A
250:    D412 C9                RET
```

内容を確認すれば一目瞭然，うまくいきましたね，B，C，Dレジスタの初期値をいろいろ変更して試してみてください。ただし，**リスト1-15**(図1-11に対応)のほうは実行前にCフラグが立っているかをXコマンドで確認しないと「あれ，89Hになった」なんてことになります。

リスト1-15

```
MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                    ORG     0D400H
110:    D400 0644               LD      B,44H
120:    D402 0E44               LD      C,44H
130:    D404 1644               LD      D,44H
140:    D406 78                 LD      A,B
150:    D407 17                 RLA
160:    D408 47                 LD      B,A
170:    D409 79                 LD      A,C
180:    D40A 17                 RLA
190:    D40B 4F                 LD      C,A
200:    D40C 7A                 LD      A,D
210:    D40D 17                 RLA
220:    D40E 57                 LD      D,A
230:    D40F C9                 RET
```

3.(HL)という書法に慣れよう

さて，前哨戦最後のとりではカッコつきのペア・レジスタです。これまでにあげた例題のリストにはこの書法を意図的に使っていません。これに慣れておけばプログラム上の表現が見やすく，書きやすくなります。

(HL)などのカッコつき表現は，ペア・レジスタだけに限りません。カッコの中にはアドレスを表わす数字やラベルも入れられます。

下の２つの命令を見てください。

①　LD　HL，5000H

②　LD　HL，(5000H)

ちがいがわかりますか？　①の場合，第２オペランド（カンマの次のオペランド）は単なる４桁の16進数です。この命令が使われたプログラム中で将来この数字が定数として機能するのか番地を表現するのかはまだわかりません。ところが②の場合は，カッコの中に収められた5000Hは紛れもないアドレスだと断言できます。

つまり，①でHLに入るものと②で入るものとにはその意味するところはもちろん，値自体のちがいがあるかもしれないのです。①でHLにロードされる値は16進の5000H，②ではメモリの5000H番地と5001

H番地の内容です。このとき，メモリの5000H，5001H番地にはどんな数が入っているか断言できません。どちらの番地ともゼロが入っているかもしれないし，偶然，5000Hという数字が入っているかもしれません。

結局，①は第1オペランドに第2オペランドの値が直接ロードされ，②は第2オペランドのカッコ内に書かれたアドレスのメモリの内容が第1オペランドにロードされます。その意味で，②のカッコ内は必ずアドレスを表現していると言えるのです。もちろん，そのアドレスのメモリ内容が定数を表わすものなのか，再び何らかのアドレスを指し示すものなのかはわかりません。

具体的な話に移りましょう。①は，一般的に「HLに5000Hをロードする」と解されており，詳しくはHレジスタに50H，Lレジスタに00Hが入ります。一方，②は「HLに5000H番地と5001H番地の2バイトのメモリの内容をロードする」と解し，詳しくは5000H番地のメモリ内容をLレジスタに，5001H番地のメモリ内容をHレジスタに入れます。レジスタとアドレスのこのような対応のしかたは一見逆のように見えますので注意してよく覚えておきましょう。HとLはそれぞれHighとLowと考えると覚えやすいです(HL以外でも同様)。

リスト1-16

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                    ORG     0D400H
110:    D400 2111D4             LD      HL,REGI
120:    D403 77                 LD      (HL),A
130:    D404 23                 INC     HL
140:    D405 78                 LD      A,B
150:    D406 77                 LD      (HL),A
160:    D407 23                 INC     HL
170:    D408 79                 LD      A,C
180:    D409 77                 LD      (HL),A
190:    D40A 23                 INC     HL
200:    D40B 7A                 LD      A,D
210:    D40C 77                 LD      (HL),A
220:    D40D 23                 INC     HL
230:    D40E 7B                 LD      A,E
240:    D40F 77                 LD      (HL),A
250:    D410 C9                 RET
260:
270:    D411            REGI:   DEFS    5
```

ここで「①は第2オペランドに書かれている5000Hだけがロードに関係しているのに，どうして②は5001Hまで出てくるの？」という疑問を持った人もいることでしょう。ここらへんをよく理解しておかないと，あとで混乱の元になりますよ。

答は以下の通りです。HLなどのペア・レジスタは16ビットの大きさを持ちます。Hだけなら8ビットですね。一方，16進数は2桁で8ビット，5000Hなどの4桁なら16ビットです。だから，①のようだと16ビットのHLレジスタに16ビットの(16進数4桁の)数が入り，ちょうどよいのです。しかし，②だとHLは16ビットなのにメモリ5000H番地の内容は1バイト，すなわち8ビットでしかありません。そこで必然的に，5001H番地までがつき合うのです。

②に関するこの説明は，ひょっとすると的外れかもしれません。つまり，第1オペランドが16ビットだから第2オペランドがつき合うのでなく，単に第2オペランドに冗長な書式——(5000H＆5001H)などの——を置きたくなかっただけなのかもしれないのです。これはCPUの(正確にはアセンブラの)開発者が，ニモニックの使用者に対して便宜を図ってくれたと理解しましょう。

説明が長くなってしまいました。ここでの例題を**リスト1-16**に示します。これはこのプログラムの実行を開始した時点でのA，B，C，D，Eレジスタの内容をメモリのREGIと名前のついた番地から5バイトにこの順で書き込むものです。110行でHLレジスタにREGIの番地をセットし，120行でその番地にAレジスタの内容をロードします。次に130行でHLを＋1し，140，150行でBレジスタの内容をロードします。150行の第2オペランドにBレジスタを直接書かないのは，そのような命令がZ80に存在しないからです。ない命令に気をつけましょう。もし知らずに書くとアセンブラから怒られます。160行以降も同じことの繰り返しです。このプログラムを実行する前にモニタのXコマンドでレジスタの内容を確認し，実行後はモニタのDコマンドでメモリのD411H番地(REGI)から5バイトがレジスタの内容と同じになっているかを見てみてください。

なお，カッコの中に書く番地やレジスタを，「アドレスを指し示すもの」という意味で「ポインタ」と呼びます。ポインタを使った書法はロード命令だけにとどまらず，ジャンプ命令，演算命令などに幅広く普及しています。

2章 覚えてしまおう マシン語の定石

基礎知識の次はちょっと高級な組み立てを覚えよう。どんなプロでもこの山を越えるのには苦労したんだ。これだけの技術を体得すれば，60行くらいのマシン語プログラムはすぐ書ける。実践的基礎テクニック集。

覚えてしまおうマシン語の定石

1 ブロック転送の使いかた

さあ，定石にはいります。まずは不思議なブロック転送命令から。

ブロック転送命令には4種類あります。うち2種は決められたバイト数だけ自動的に繰り返すもの，残り2種はユーザーが繰り返させるものです。それらの説明の前に，ブロック転送という語が表わす意味から紹介しましょう。

1. ブロック転送とは

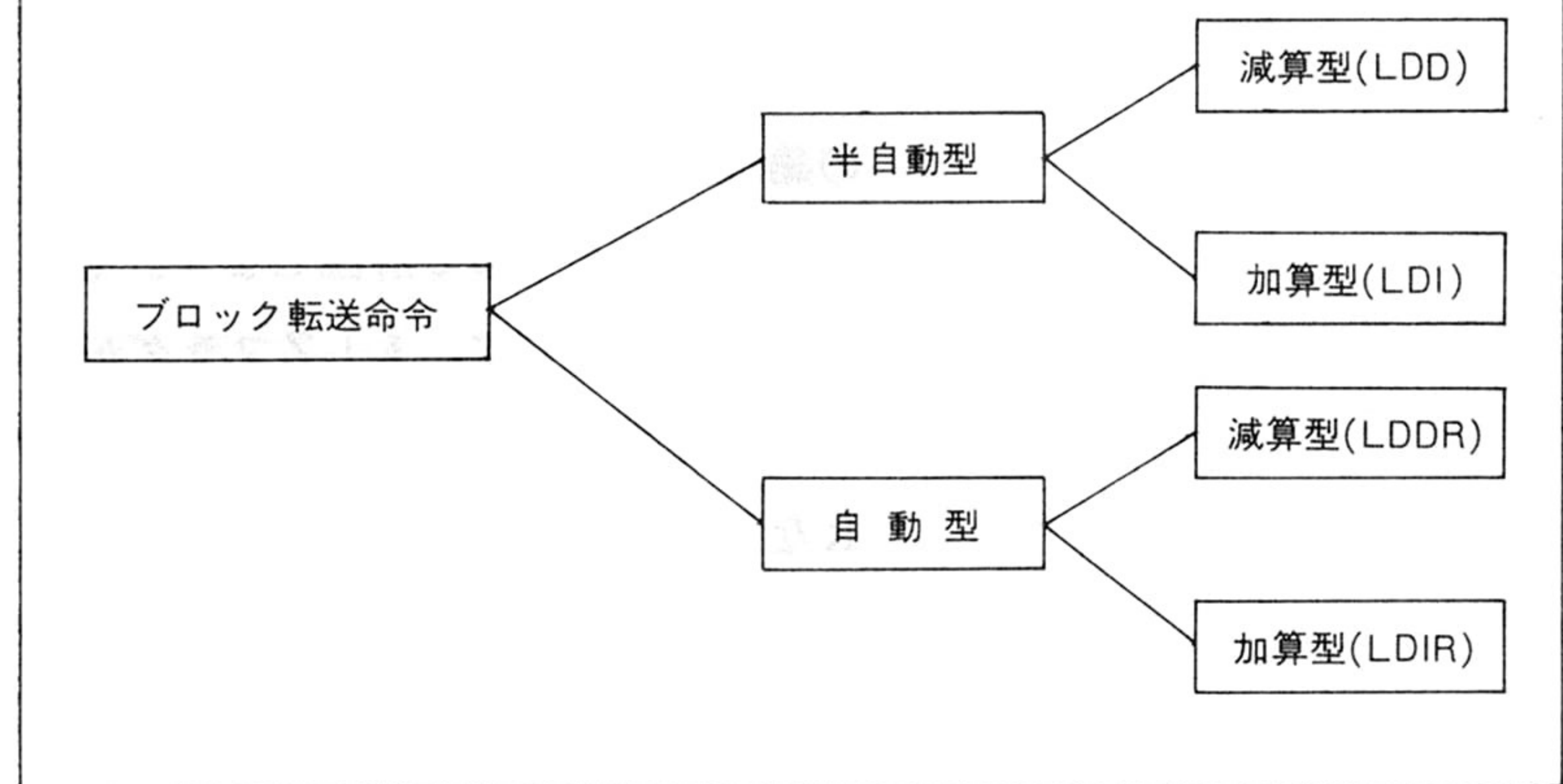
「ブロック転送」とは，ブロック，すなわちメモリの固まりを元あった番地からある番地へそっくりそのままコピーすることです。転送先のメモリ・ブロックの範囲は転送前と重なっていてもかまいません。ブロック転送命令は1命令でこれができるのです。

ブロック転送命令は，実行前にBC, DE, HLの各ペア・レジスタを設定する必要があります。DEとHLはどちらもポインタで，ＤＥには転送先の，ＨＬには送り側のブロック先頭アドレスを入れておきます。BCはカウンタで，転送するバイト数を入れます。BCレジスタは16ビットなので，64Kバイトまでの大きさのブロックを転送できることになります。

これら3つのペア・レジスタを設定した後，ブロック転送命令を実行します。ブロック転送命令4種のうち2種類は自動型で，ブロック転送命令を実行した直後からＢＣレジスタのカウンタがゼロになるまで転送しつづけます。もしカウンタが400バイトを示していれば，MSXでは約2.5m秒(25／1000秒)間はこの命令にかかりっきりになります。

自動型でないものもあります。半自動とでもいいましょうか。これもBC, DE, HLレジスタを事前に設定しなければなりませんが，1バイト転送するごとに次の命令に進みます。BCのカウンタがゼロになったかどうかはP／Vフラグを見て人間が判断します。これが半自動といわれる理由です。どうしてわざわざ半自動型があるのかは，のちに取り上げます。

ブロック転送命令には自動型と半自動型の区別のほかに，転送順の区別があります(**表 2-1**)。加算型というのは，ポインタのDEとHLを＋1しながら転送する方式，減算型はポインタを－1しながら転送する方式です。これらの使い分けもまた別項で取り上げます。

表2-1　ブロック転送命令

2. ブロック転送命令の機能

ブロック転送命令は1命令でいろいろなことをしています。つまり，HLレジスタをポインタとするメモリの内容をDEレジスタをポインタとするメモリにロード，そしてBCレジスタの内容を－1，もしこれでBC＝0になればブロック転送は終了で次の命令に進み，そうでなければ次の1バイトの転送に進む，といったぐあいです。LDIR命令を例として，これらの動作を等価的にプログラムにしたものが**リスト 2-1**です。D400H番地から13Hバイト分をD450H番地からに転送しています。D400H番地から19バイトは，実はこのリストのオブジェクトです。つまり自分自身を転送しているわけです。

リスト2-1

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    D400                    ORG      0D400H
110:    D400 2100D4             LD       HL,0D400H
120:    D403 1150D4             LD       DE,0D450H
130:    D406 011300             LD       BC,0013H
140:    D409 7E         NEXT:   LD       A,(HL)
150:    D40A 12                 LD       (DE),A
160:    D40B 23                 INC      HL
170:    D40C 13                 INC      DE
180:    D40D 0B                 DEC      BC
190:    D40E 78                 LD       A,B
200:    D40F B1                 OR       C
210:    D410 20F7               JR       NZ,NEXT
220:    D412 C9                 RET
```

190～210行が要注意です。ここは180行で－1したBCレジスタがゼロになったかどうかを判定しているところで，ミソは200行のOR命令です。この命令は論理演算命令といって，AレジスタとCレジスタとの論理和をとります。詳しいことはのちに「論理演算命令」について取り上げるところで解説します。ともかくBC＝0ならこうするとZフラグが立つのです。もしZフラグが立っていなければ(BC＝0でなければ)210行でNEXTにジャンプし，次のバイトの転送にうつります。「ORなんてややこしいもの使わなくても，Zフラグを見ればいいじゃないか」と思う人はあとの「ループ」の項をごらんください。

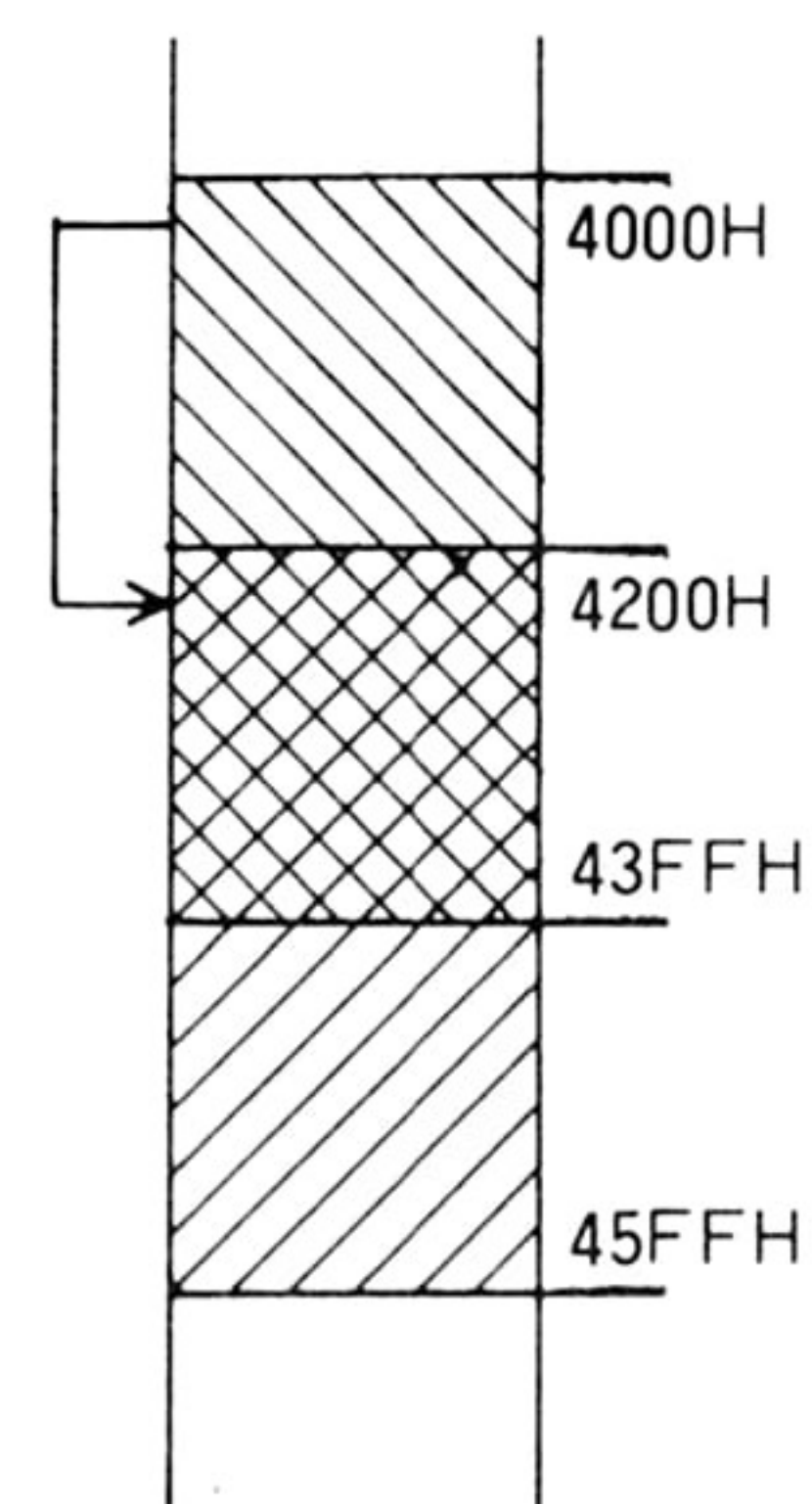
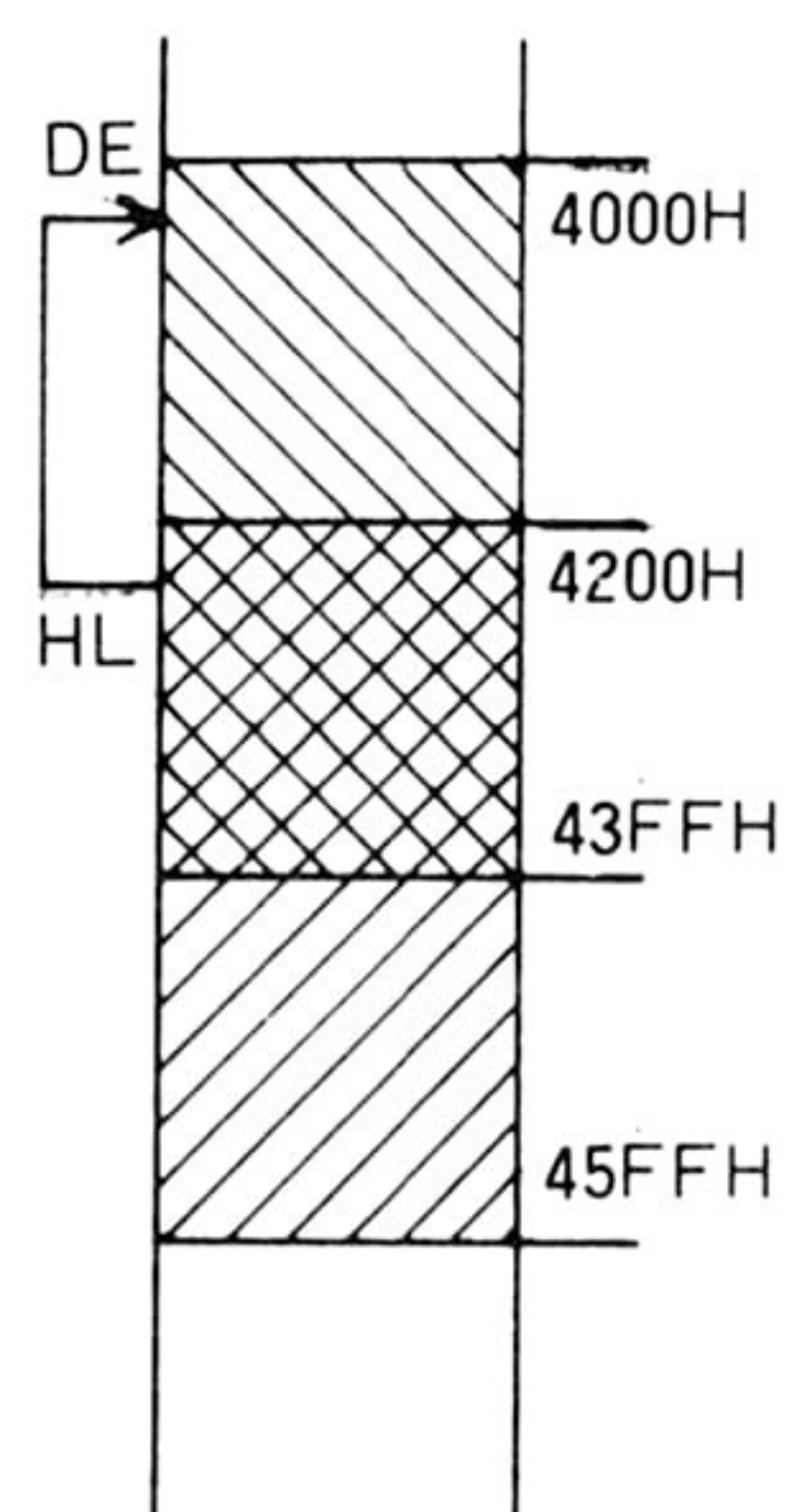
3.LDIRとLDDRの使い分け

どうせ転送してしまうんなら上位からでも下位からでも同じ，と思いませんか。確かに転送されるメモリ・ブロックと転送先のメモリ・ブロックとが完全に独立しているときは，どちらをとっても同じです。ところが1バイトでもブロックの重なりがあると，そうはいきません。

図2-1

図2-1を見てください。いまここで，4000H～43FFH番地までのブロック400Hバイトを転送されるブロック，転送先は4200H番地からとしてみました。このようなとき，LDIRを使うとどうなるでしょう？　LDIRは加算型ですからポインタはどちらも両ブロックの先頭に初期設定しておかなくてはなりませんね。つまりこの場合，BC＝400H，DE＝4200H，HL＝4000Hとなります。こうしてLDIR命令を実行すると4000H番地の内容が4200H番地に，(4001H)→(4201H)に，というようにどんどん転送されていきます。ところが送り手(HL)が4200H番地まで進むと？　そう，4200H番地の内容は最初に4000H番地から転送されたものでしたね。ということは，最初のメモリブロックの状態と異なる状態で転送されてしまいます。

図2-2

そこでLDDR命令を使ってみましょう。初期設定はBC＝400H，DE＝45FFH，HL＝43FFHです。これを実行すると，(43FFH)→(45

FFH)，(43FEH)→(45FEH)，……，(4200H)→(4400H)，……，(4000H)→(4200H)となって，何の不都合も起きません。転送先を指すポインタが43FFHを示すころには，すでにここの元のデータは45FFHに転送済です。かまわず転送してしまってもいいのです。

逆に，**図2-2**のようなときはLDIR命令を使うのが適当です。LDDR命令を使うとブロックの内容を壊してしまいます。

このような使い分けを見破るコツは，転送元のブロックの両端（**図2-1**なら4000Hと43FFH）のうちで転送先ブロックと重なっていないほうを転送の先頭アドレスとすることです。

4.LDIとLDDを使う

加算型と減算型があることの理由はわかりました。ではなぜ自動型と半自動型があるのでしょうか。以下のような理由が考えられます。

①半自動型なら，1バイト転送してから次の1バイトを転送するまでに何か別の処理をおくことができる。

②転送される側のデータが連続でなくてもよい。

①は，時によって非常に大きな利用価値をもちます。もし,「転送されるデータのうち値がゼロのものがあったら転送しない」ということにすれば，**リスト2-2**のようになります。**リスト2-2**のオブジェクトにはゼロの値を持つものが3バイトあるので，23バイトのうちゼロでない20バイトだけが詰めて転送されるわけです。p.36の**リスト2-3**に双方の結果を示すメモリダンプを示しますので，比べ見てください。

リスト2-2

```
  MSX Self Assembler  Rev 1.0    PAGE    1


 100:   D400                     ORG     0D400H
 110:   D400 2100D4              LD      HL,0D400H
 120:   D403 1150D4              LD      DE,0D450H
 130:   D406 011700              LD      BC,0017H
 140:   D409 EDA0      NEXT:     LDI
 150:   D40B E216D4              JP      PO,OWARI
 160:   D40E 7E                  LD      A,(HL)
 170:   D40F FE00                CP      0
 180:   D411 20F6                JR      NZ,NEXT
 190:   D413 23                  INC     HL
 200:   D414 18F3                JR      NEXT
 210:
 220:   D416 C9        OWARI:    RET
```

リスト2-3

```
D400 21 00 D4 11 50 D4 01 17 : 42
D408 00 ED A0 E2 16 D4 7E FE : D5
D410 00 20 F6 23 18 F3 C9    : 0D
D450 21 D4 11 50 D4 01 17 ED : 2F
D458 A0 E2 16 D4 7E FE 20 F6 : FE
D460 23 18 F3 C9             : F7
```

リスト2-2の140行にLDI命令があります。LDIやLDDの半自動型ブロック転送命令は，1バイトを転送したあとカウンタのBCレジスタを－1し，次命令に進みます。次命令ではBCレジスタがゼロになったかどうかを判定してやらねばなりません。ゼロになったかどうかの情報はP／Vフラグに入っています。ふつう，ゼロかどうかはZフラグで判断しますが，LDI，LDD命令だけはP／Vでゼロを判断するのです。これによって，さきの①のように，繰り返しの途中でブロック転送以外の処理を置くことができるのです。**リスト2-2**では180行でZフラグを使っていますね。もしブロック転送カウンタBC＝0をZフラグで判断していたとしたら，とても170～180行のようなことはできません。これもまた，CPU設計者の配慮といえるでしょう。

ともかく，BC＝0かどうかはP／Vフラグで判定します。P／Vフラグを使った条件ジャンプはJP PO,～とJP PE,～の2つで，POのほうはP／Vフラグが立っていない(BC＝0)，PEはP／Vフラグが立っている(BC≠0)を表わしています。この判定は150行でしており，もしBC＝0ならOWARIに飛んでリターン，BC≠0なら次の転送データがゼロかどうかの判定にうつります。170～180行でそれを判定し，もしBC＝0なら190行で転送元のカウンタを1バイト強制的に進めます。ゼロでなければ次のメモリの転送(NEXT)に飛びます。

このプログラムの欠点は，もし転送しようとする元ブロックの先頭にゼロがあったら，それは転送されてしまうということです。転送しようとするデータがゼロかどうかを判定する170，180行を通るときには，もうすでに1バイトを転送してしまっているからです。これを防ぐための改造法は，皆さんが考えてみてください。

さて，②はおわかりでしょうか。自動型のブロック転送命令は，決められたメモリ・ブロックをすきまなく転送します。転送されるデータがメモリ上に連続的に置かれているならこれでよいのでしょうが，たとえば図2-3のような配置のデータをつめて転送したいときは，も

う自動型は使えません。半自動型ならこのような場合にも対処できます。実際に2バイト間隔で並べられたデータをすきまなく詰め直すプログラムがp.38の**リスト2-4**です。実行前のデータはD450H番地から16バイトの中に2バイト間隔で6バイト(**図2-4**参照)分です。これを同じD450H番地から6バイトに並べかえるわけです。実行前後のメモリ・ダンプも掲載しておきました。

特に複雑なことはしていません。140行で1バイト転送してから，150，160行で転送元のポインタを2つ進めています。170行でカウンタのBCレジスタがゼロになったかどうか判定し，PE(BCレジスタがゼロでない)ならNEXTで次の1バイトにとりかかります。

200行からは転送されるデータの初期設定です。200行で以後はD450H番地からオブジェクトを発生するようにアセンブラに指示しています。このように，ORG擬似命令は1つのプログラム中で何度でも設定しなおせます。210行のDATA以降はDEFB擬似命令とDEFS擬似命令の嵐です。DEFBでその番地の値を設定し，DEFSで2バイトの間隔を開けています。ちなみに，DEFS命令は単に アドレスをオペランドの分だけ進めるだけで，飛ばされた番地の内容は変えません。

図2-3

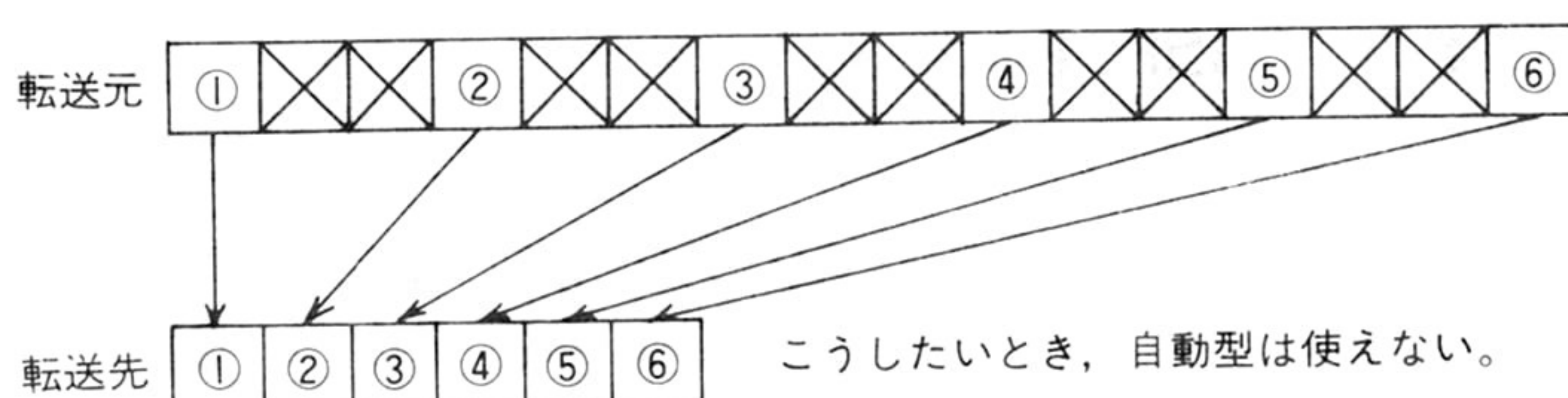
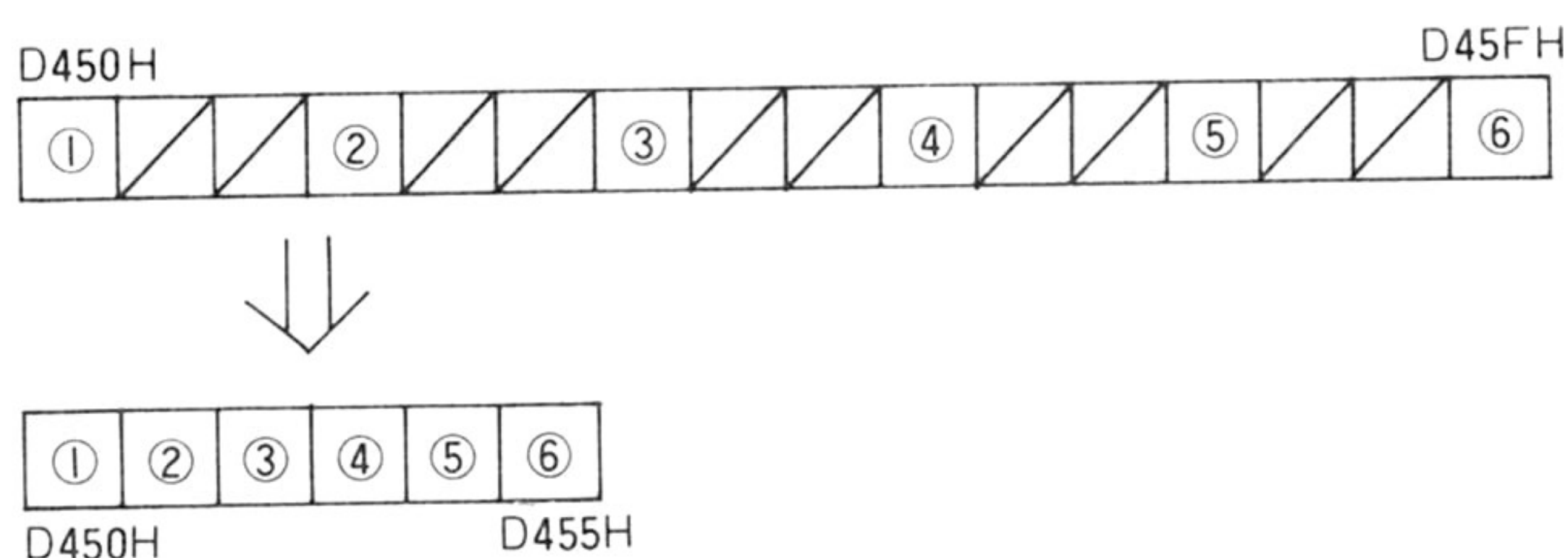



図2-4

リスト2-4

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                    ORG     0D400H
110:    D400 2150D4             LD      HL,DATA
120:    D403 1150D4             LD      DE,DATA
130:    D406 010600             LD      BC,6
140:    D409 EDA0       NEXT:   LDI
150:    D40B 23                 INC     HL
160:    D40C 23                 INC     HL
170:    D40D EA09D4             JP      PE,NEXT
180:    D410 C9                 RET
190:
200:    D450                    ORG     0D450H
210:    D450 01         DATA:   DEFB    1H
220:    D451                    DEFS    2
230:    D453 02                 DEFB    2H
240:    D454                    DEFS    2
250:    D456 03                 DEFB    3H
260:    D457                    DEFS    2
270:    D459 04                 DEFB    4H
280:    D45A                    DEFS    2
290:    D45C 05                 DEFB    5H
300:    D45D                    DEFS    2
310:    D45F 06                 DEFB    6H
```

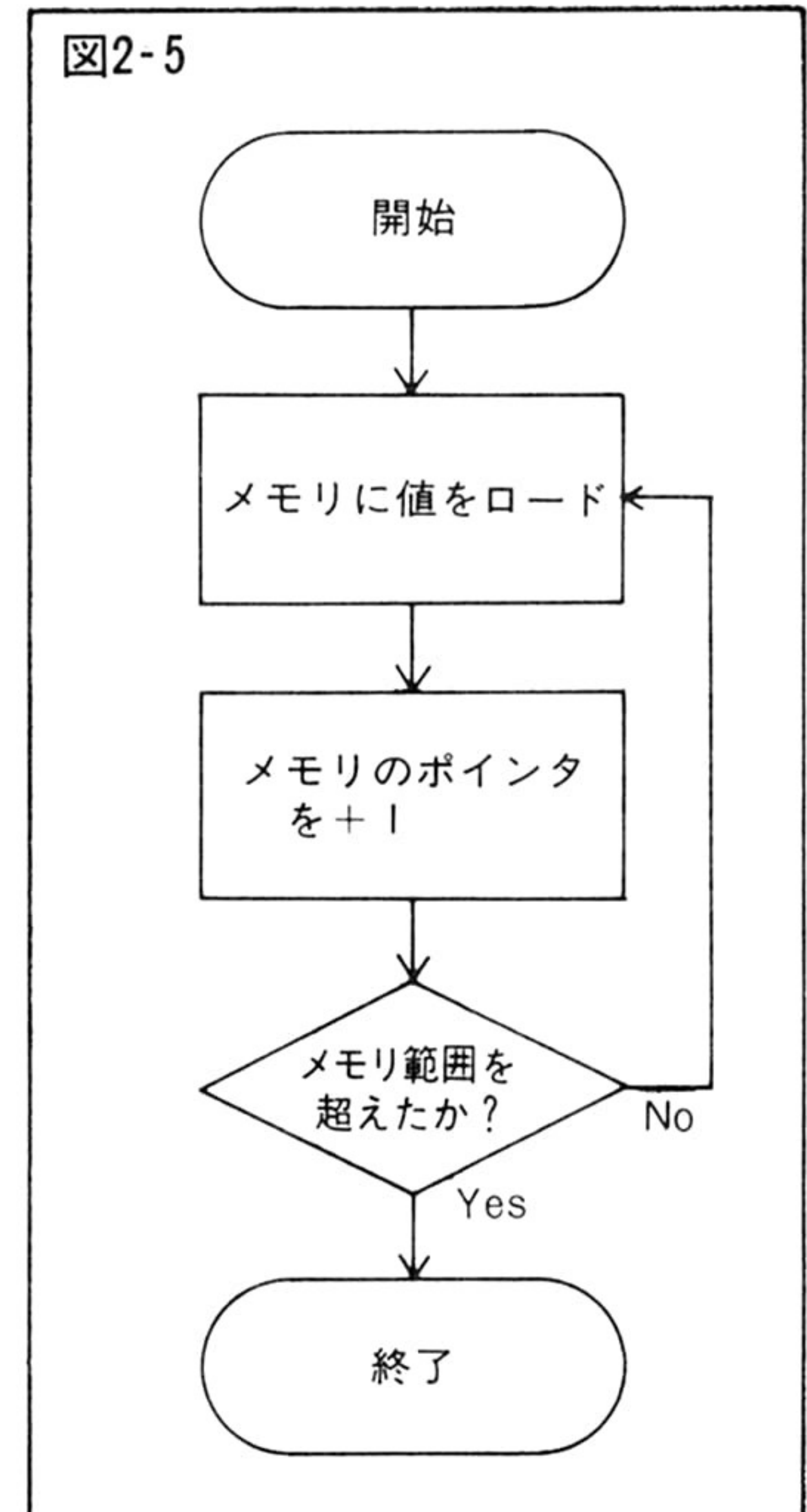
```
D450 01 00 00 02 00 00 03 00 : 06
D458 00 04 00 00 05 00 00 06 : 0F

D450 01 02 03 04 05 06 03 00 : 18
D458 00 04 00 00 05 00 00 06 : 0F
```

5.フィル・メモリのプログラム

最後に，ブロック転送命令のちょっと変わった使いかたを紹介しておきましょう。例として「フィル・メモリ」を取り上げました。

「フィル・メモリ」とは，指定した範囲のメモリを指定した値で埋める(Fill：満たすの意）動作を指すことばです。たとえば値として1を指定すれば，指定した範囲のメモリ内容はすべて1に書きかえられるのです。

ふつうなら，フィル・メモリをするプログラムは図2-5のような考えかたでつくられます。この前に指定したいメモリ範囲や埋めたい値を入力すればOKです。

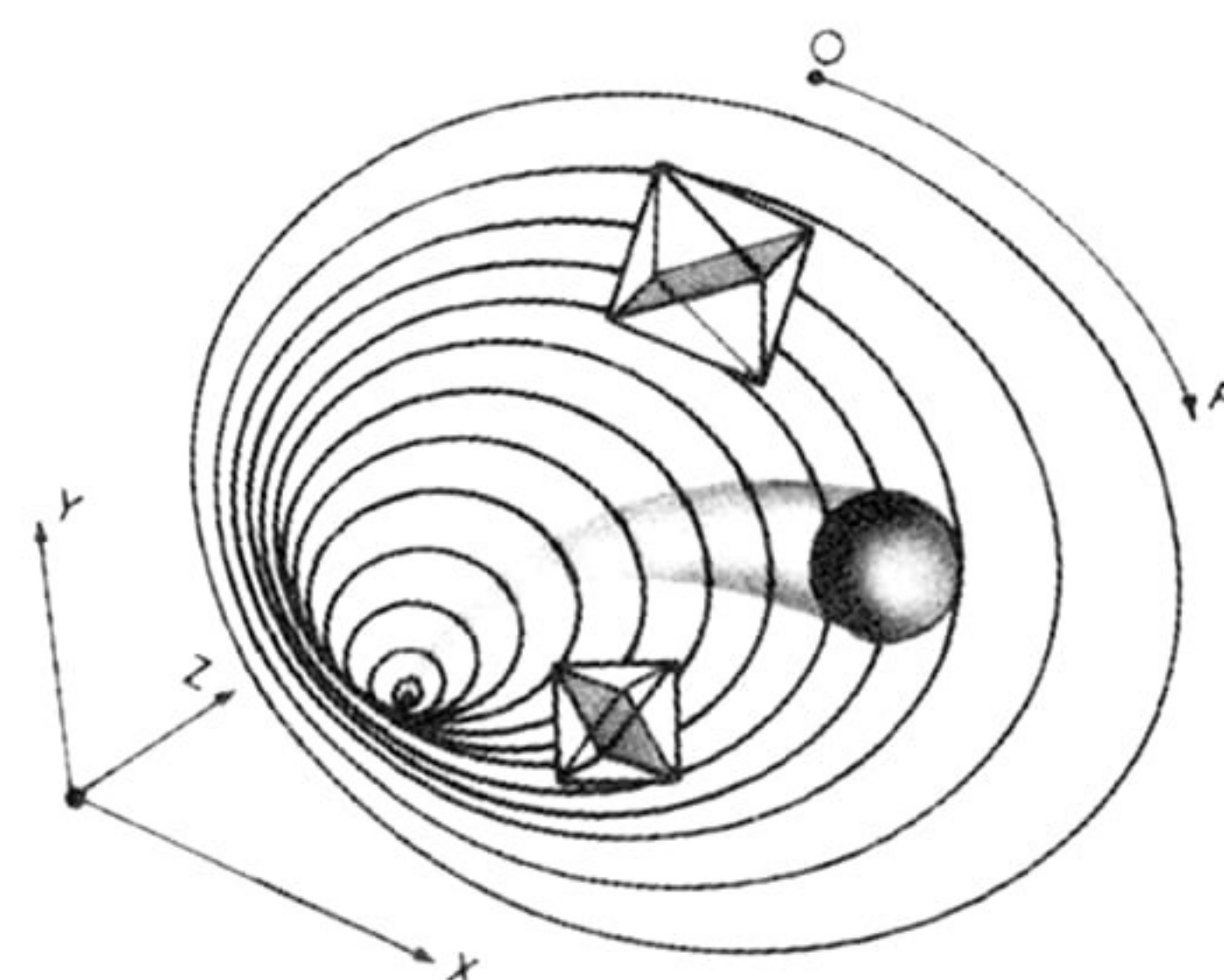
図2-5

しかし，このようなことをしないでも，ブロック転送命令でもっと簡潔に同じ動作をするプログラムが書けるのです。**リスト** 2-5を見てください。これは，D430H番地からD49FH番地までをDATAと名前のついた行の値で埋めるのです。実行前にモニタのDコマンドでこの範囲のメモリ内容をのぞいておいて，実行後にその効果をたしかめてください。DATA行のDEFB命令のオペランドの値を適当に変えると楽しめます。フィルするメモリ範囲をD500H以上にするとアセンブラを壊してしまいますからご注意を。また，D4FFH付近もメモリの使いかたの状態によっては，書きかえると暴走しがちです。フィル・メモリはD4AFHあたりまでにしてください。

参考までに，**リスト** 2-6にLDIRを使わないで同じメモリ範囲をフィルするプログラムをあげておきました。意図に反して(？)**リスト**2-6のほうがブロック転送を使ったものよりも短いオブジェクトになってしまいました。でも，①ブロック転送を使ったもののほうが実行時間が短い(約3割減)，②**リスト** 2-6ではカウンタが8ビットなので256(100H）バイトまでの転送しかできないが，ブロック転送を使えばカウンタは16ビットで65536(10000H)バイトまで転送可能，の2つの利点があります。

リスト2-5

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    D400                       ORG      0D400H
110:    D400 2130D4                LD       HL,0D430H
120:    D403 3A10D4                LD       A,(DATA)
130:    D406 77                    LD       (HL),A
140:    D407 1131D4                LD       DE,0D431H
150:    D40A 016F00                LD       BC,0D49FH-0D430H
160:    D40D EDB0                  LDIR
170:    D40F C9                    RET
180:
190:    D410 99         DATA:      DEFB     99H
```

リスト2-6

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    D400                       ORG      0D400H
110:    D400 2130D4                LD       HL,0D430H
120:    D403 0670                  LD       B,0D4A0H-0D430H
130:    D405 3A0ED4                LD       A,(DATA)
140:    D408 77         NEXT:      LD       (HL),A
150:    D409 23                    INC      HL
160:    D40A 05                    DEC      B
170:    D40B 20FB                  JR       NZ,NEXT
180:    D40D C9                    RET
190:
200:    D40E 55         DATA:      DEFB     55H
```

覚えてしまおうマシン語の定石

2 ループの作りかた

つづいて，ループ（繰り返し）処理に関する定石について書いてみます。

人間にとってめんどうな作業をコンピュータにやらせようというのですから，プログラムに繰り返しを含む手つづきが含まれるのは当然です。しかも，プログラムの大きさや速度をよりコンパクトにまとめようとすれば，プログラム中のループの役割はますます大きくなるでしょう。

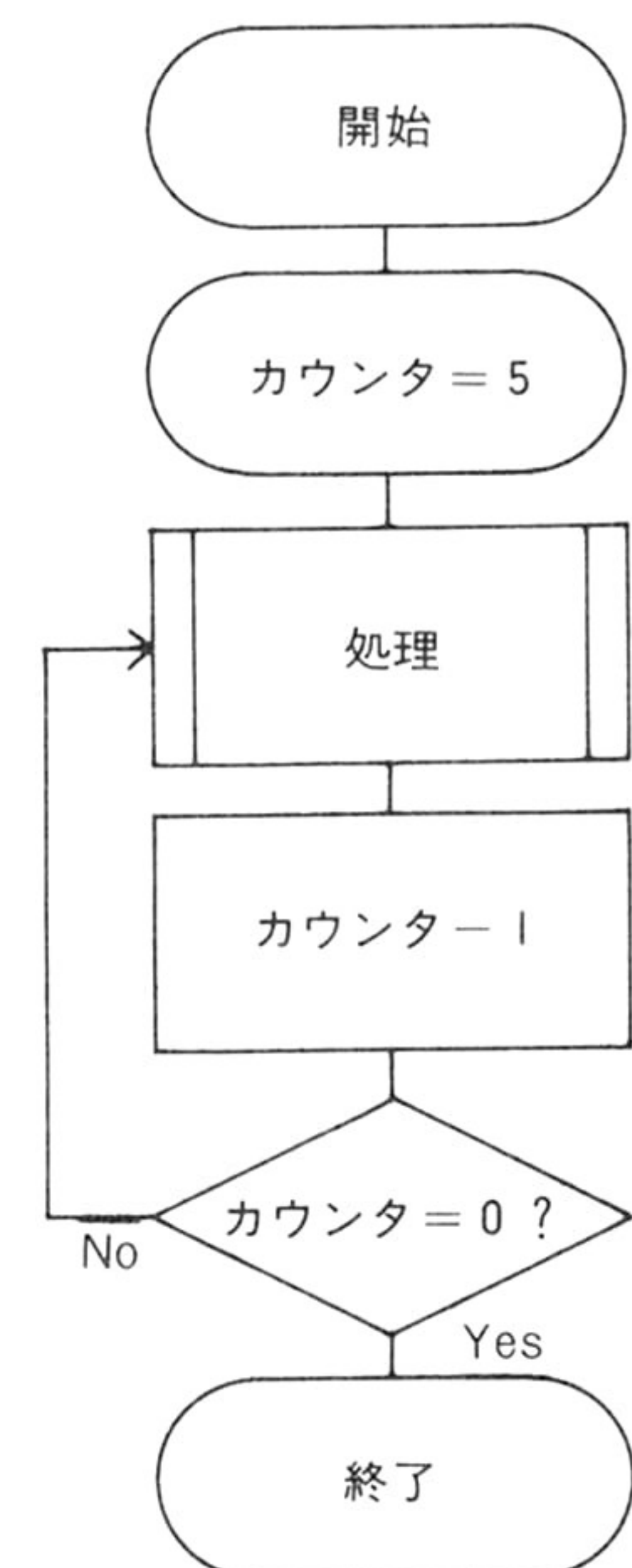
手順を制御するしくみ「制御構造」としてのループは，だいたい次の3つに分けられます。

①回数を決めたループ

②AからBまで，Cきざみで繰り返すループ

③ある条件が成立するまで（あるいは不成立まで）繰り返すループ

①は，ループをまわる回数を数えるカウンタ(ループ・カウンタ)の上限をあらかじめ設定してからループを開始するのです。簡単なフローチャートで表わせば**図2-6**のようになります。この図の場合，カウンタは5回で，ループ内での処理を1回終えるごとにカウンタをデクリメントし，カウンタがゼロになるまで繰り返します。カウンタのデクリメントやゼロかどうかの判定などはマシン語レベルでも簡単に実現できるので，回数が最初からわかっているならこの方法が楽です。入力された文字を5回プリントして戻るプログラムをp.42の**リスト2-7**にあげておきます。Eレジスタをループ・カウンタにしています。160行のオペランドの数を変えればもっとたくさん文字をプリントさせられます。180行でループ・カウンタをデクリメントし，190行でゼロかどうかを判断します。も

図2-6

しまだゼロでなければLOOPにジャンプし，繰り返します。150行のBIOSコールはキーボードからの1文字入力，170行は画面への1文字出力です。

リスト2-7

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                     ORG      0D400H
110:
120:    009F =         CHGET     EQU      009FH
130:    00A2 =         CHPUT     EQU      00A2H
140:
150:    D400 CD9F00              CALL     CHGET
160:    D403 1E05                LD       E,5
170:    D405 CDA200    LOOP:     CALL     CHPUT
180:    D408 1D                  DEC      E
190:    D409 20FA                JR       NZ,LOOP
200:    D40B C9                  RET
```

②は，①をもう少し柔軟にしたループです。ループ・カウンタの値は必ずしも回数そのものでなくてもよく，ループ・カウンタの増分は1でなくてもよいのです。増分が負である場合もあります。言ってみればBASICにおけるFOR文の FOR A TO B STEP C のようなものです。Aをループの回数の値そのもの,Bを1，Cを－1とすると①と同じことになります。

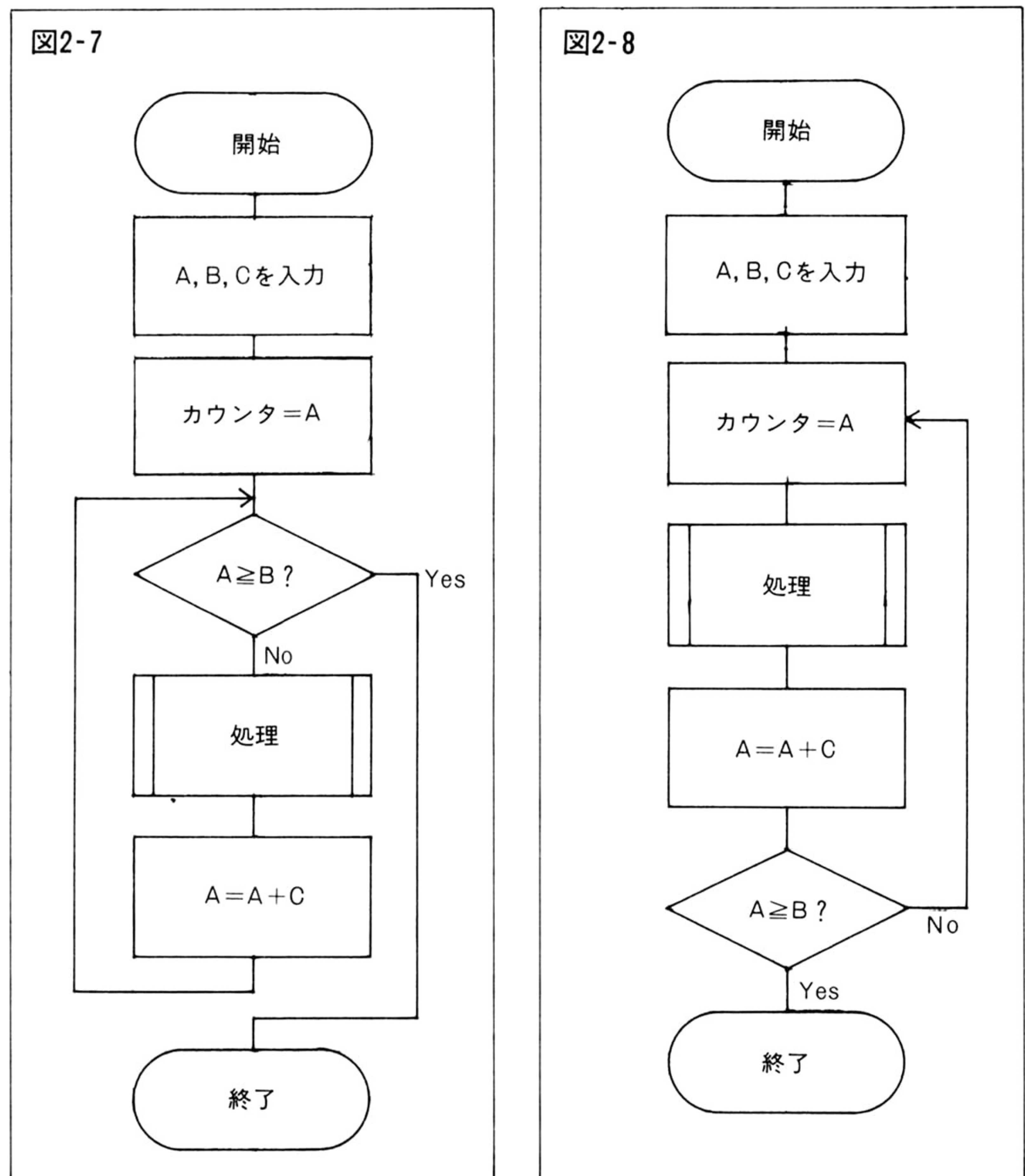
この種のループは，決められた回数を繰り返したかどうかの判定で「カウンタ＝0？」の手法が使えません。つねにループ・カウンタと終了条件のBの値とを比較する必要があるのです。また，ループ・カウンタの増減もインクリメントやデクリメント1発で済ませられるとは限りません。ここらあたりが①のループとちがって複雑なところです。

フロチャートにすると**図2-7**のようになります。**図2-8**も働きは同じですが，こちらのほうだともし最初からAが終了条件を満たしていても（A≧Bでも)，処理を1度だけ通過してしまいます。それは条件の判定が処理のあとにあるからです。また，**図2-7,8**とも増分Cが負の場合は考慮してありません。

図2-7を具体化したプログラムが**リスト2-8**です。これは0000H番地から00EEH番地までのうち，偶数番地のメモリ内容で最大の値をAレジスタに，その番地をDEレジスタに入れて戻るプログラムです。偶数

とは2で割って余りがゼロのもののことで，0000H，0004Hはもろん，000CH，000EHなども偶数といえます。0000Hから00EEHまでのうちの偶数番地は119個あります。

図2-7

図2-8

リスト2-8

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:     D400                    ORG      0D400H
110:     D400 210000             LD       HL,0
120:     D403 3E00               LD       A,0
130:
140:     D405 4F        LOOP:    LD       C,A
150:     D406 7D                 LD       A,L
160:     D407 FEF0               CP       0F0H
170:     D409 79                 LD       A,C
180:     D40A 280D               JR       Z,RTN
190:
200:     D40C 46                 LD       B,(HL)
```

```
210:    D40D 23                    INC      HL
220:    D40E 23                    INC      HL
230:    D40F B8                    CP       B
240:    D410 30F3                  JR       NC,LOOP
250:    D412 78                    LD       A,B
260:    D413 54                    LD       D,H
270:    D414 5D                    LD       E,L
280:    D415 1B                    DEC      DE
290:    D416 1B                    DEC      DE
300:    D417 18EC                  JR       LOOP
310:
320:    D419 C9         RTN:       RET
```

少々長いリストになってしまいました。110行でループ・カウンタ(この場合は参照するアドレスが入る), 120行で結果を入れるレジスタをそれぞれクリアしています。140行から180行までは, ループ・カウンタが00EEH番地を超えたかどうかのチェックです。結果の入ったAレジスタはいったんCレジスタに退避させ, ループ・カウンタの下2バイト(Lレジスタ)をAに入れてF0Hになっているかどうかを調べています。条件はEEHまでなのになぜF0Hで比較しているかというと, このチェックにまわってくるときはすでに前のループの中で2度インクリメントされている(210, 220行)からです。それでもしループ・カウンタHLが00F0Hになっていたら, 00EEHまで済んだということですから, 180行でRTNにジャンプし, リターンします。ここでもしF0HでなくEEHと比較したとすれば, 00EEH番地の内容が最大値であるかどうかの評価の対象にならずに終わってしまいます。また, EFHと比較したとするとカウンタHLは00EFHになりえないので, 64Kバイトのメモリ空間をマタにかけた遠大な無限ループにはまってしまいます。

200行からは最大値の判定です。200行でカウンタの指すメモリの内容をBレジスタに入れ, 先にカウンタを進めておいてから230行でAレジスタと比べます。Aレジスタにはこれまでの最大値が入っているので, もしBレジスタのほうが小さいか同じならそのまま240行でLOOPに戻ります。A<Bなら新しい最大値の出現ですから, 250行でAレジスタを更新します。260行〜290行では, 最新の最大値が発見されたアドレスをDEレジスタにしまっています。DEレジスタを－2しているのは, 先にカウンタだけが2つ進められているのを補正するためです。

わかりましたか。このプログラムは, モニタで実行するときに単なるGD400↵でなく, GD400, D419↵とする必要があります。こうしな

いと，RETでBASICに戻ったときにせっかくプログラム中で得たA，DEレジスタが壊されてしまうからです。また，調べたメモリがすべてゼロだったときに限って，DEレジスタの内容はデタラメです。理由はみなさんで考えてください。

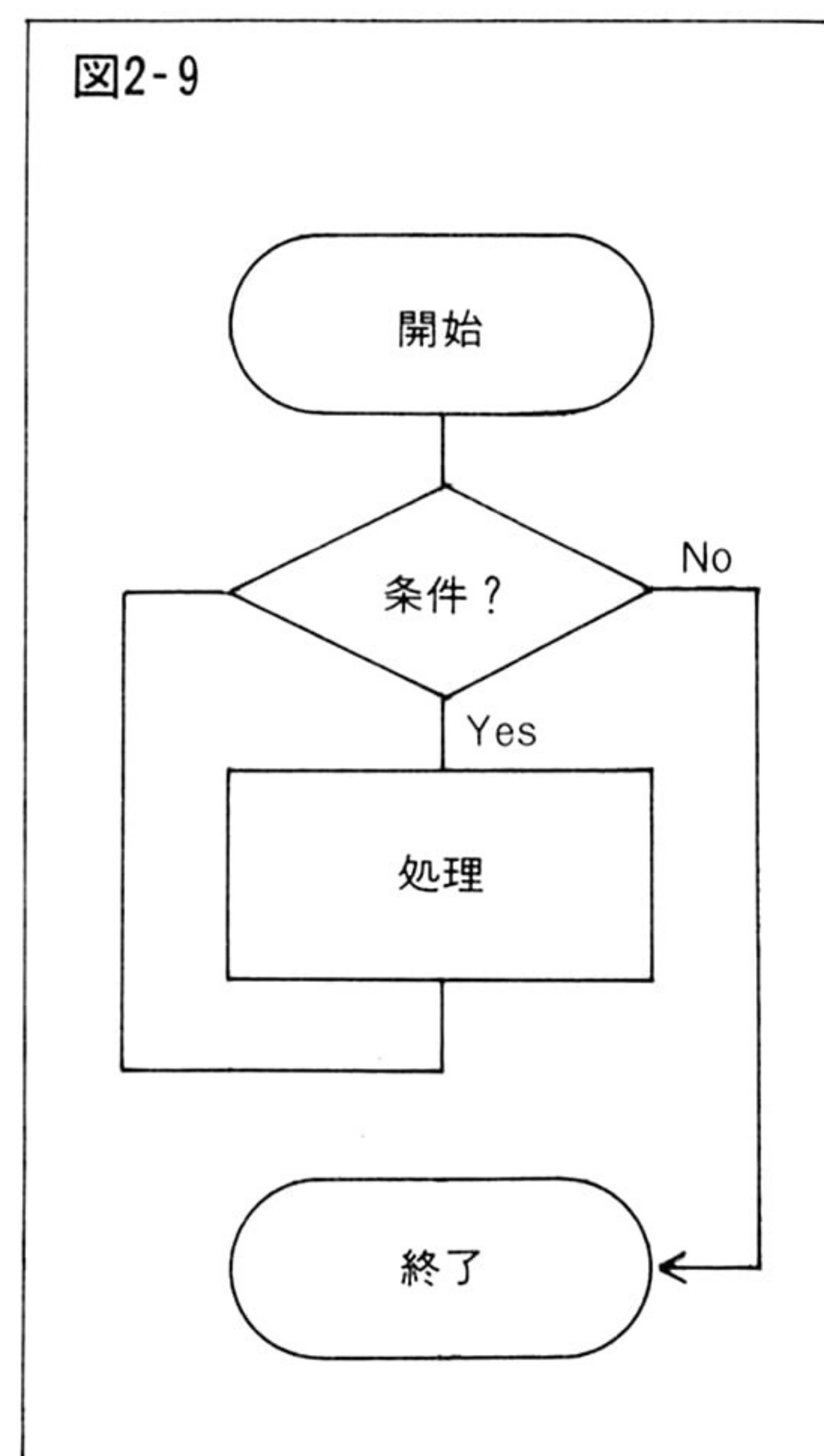
③に移りましょう。ある条件が成立あるいは不成立の間は，ある処理をしつづけるというものです。

```
1000 A$=INKEY$
1010 IF A$="" THEN1000
```

などとしてよく使われますね。これはこれまでの①，②とちがって，ループを何回まわるかの保証がありません。へたをするとプログラムは電源を切られるまでループするはめになります。

フローチャートは**図2-9**のようになります。これは「条件」が成立しているうちは「処理」を実行するというものですが，YesとNoをつけ変えれば「条件」が成立してない間だけループするというようにもできます。次の頁の**リスト2-9**はこの種のループの例です。まずキーボードからの入力を待ち，入力されたキーの文字をプリントしてループします。リターン・キーが入力されたときだけこのループから抜け出します。プログラムは短く簡単ですね。170行のリターン命令は，条件リターンです。つまり，160行の比較によってZフラグが立ったときだけ（すなわち押されたキーのコードが0DH〔リターン・キー〕だったときだけ）RETするわけです。

図2-9

ループについての概論が長くなってしまいました。つぎに各論にうつりましょう。

リスト2-9

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                     ORG     0D400H
110:
120:    009F =        CHGET      EQU     009FH
130:    00A2 =        CHPUT      EQU     00A2H
140:
150:    D400 CD9F00   LOOP:      CALL    CHGET
160:    D403 FE0D                CP      0DH      ;CR KEY
170:    D405 C8                  RET     Z
180:    D406 CDA200              CALL    CHPUT
190:    D409 18F5                JR      LOOP
```

1.8ビットのループカウンタ

まず，ループ・カウンタが8ビットレジスタのときを考えてみます。カウンタが8ビットだとループできる回数が少ない代わりに，判断が楽にすみます。

リスト2-10は，Bレジスタをループ・カウンタとして，0から255個の数をD300H番地から順に書き込むプログラムです。

110行では書き込むアドレスをHLに設定しています。120行でループ・カウンタのBレジスタに最高値のFFHを入れてから130行からのループに入っています。

ループ内ではカウンタを－1，アドレスを＋1しながらアドレスの下位1バイト(16進2桁)を次々とロードしています。130行の第2オペランドをLレジスタでなく何かほかの(たとえばCレジスタ)にしてもよいのです(カウンタとともにCレジスタも＋1すれば)。しかしここではHLの下位が00から始まるのを利用してLレジスタにしています。160行ではループ・カウンタがゼロになったかどうかを判定し，まだならLOOPにジャンプします。ゼロなら170行でリターンします。

このプログラムをアセンブル，実行後，モニタのDコマンドでD300H番地からをのぞいてみてください。実行前にも見てみるといいですね。どうですか。D300H番地から順序よく00，01，02…と並んでいるでしょう。最後のほうはどうですか。D3F0H番地にはF0H，次がF1Hになってますね。ところがD3FFH番地は？　FFHになってますか？D3FEH番地はちゃんとFEHになっているのにD3FFH番地はFFHではないですね。これはループ・カウンタがFFHから始まっているために，D300H番地からFF個，D3FEH番地までしか届かないからです。

D3FFH番地まで届かせるためにはループ・カウンタを1つ増せばいいのですが，もうFFHと最大の値になっているし……。

ところが，解決法があるのです。カウンタをFFHから始めると255回の繰り返ししかしませんが，カウンタの初期値をゼロにすればよいのです。**リスト**2-11を見てください。ループ・カウンタの初期値を120行でゼロにしていますね。こうすると「あとでカウンタがゼロになったかどうか確認するから，1回もループを実行しないうちにループから抜け出ちゃうんでは？」と思う人もいることでしょう。でも大丈夫。カウンタがゼロかどうか調べる(160行)のは，150行でカウンタを－1してからなのです。ということは，カウンタの初期値がゼロでも，160行で調べるときには150行でFFHにされているわけです。これでループを1回かせげましたね。**リスト**2-11を実行すると，ちゃんとD3FFH番地までFFHが書き込まれていることがわかると思います。

8ビットのループ・カウンタを1つだけ使うなら，256回のループがせいいっぱいです。これより多い回数のループはカウンタを16ビット

リスト2-10

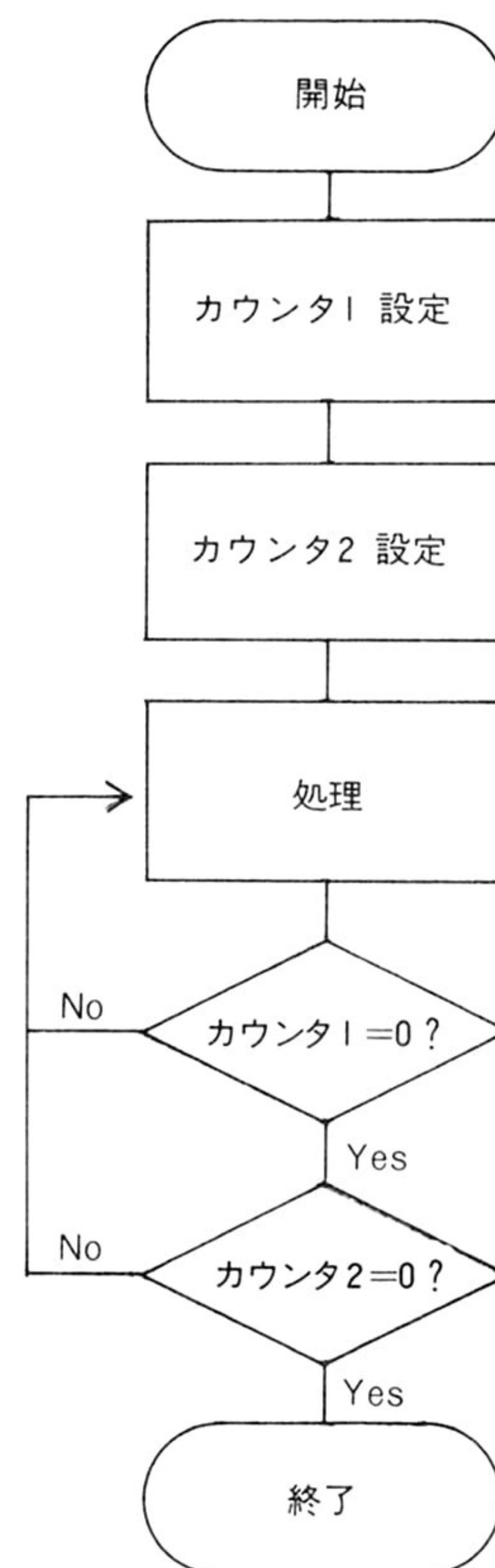
```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                    ORG     0D400H
110:    D400 2100D3             LD      HL,0D300H
120:    D403 06FF               LD      B,0FFH
130:    D405 75         LOOP:   LD      (HL),L
140:    D406 23                 INC     HL
150:    D407 05                 DEC     B
160:    D408 20FB               JR      NZ,LOOP
170:    D40A C9                 RET
```

リスト2-11

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                    ORG     0D400H
110:    D400 2100D3             LD      HL,0D300H
120:    D403 0600               LD      B,0H
130:    D405 75         LOOP:   LD      (HL),L
140:    D406 23                 INC     HL
150:    D407 05                 DEC     B
160:    D408 20FB               JR      NZ,LOOP
170:    D40A C9                 RET
```

にするか，8ビットのカウンタを複数個組み合わせるしかありません。16ビットのカウンタに関しては次項にまわして，8ビット・カウンタを組み合わせる方法をここで取り上げます。

概略のフローチャートを図2-10に示します。ここではカウンタを2重にしていますが，必要であれば何重にもできます。ただし，カウンタはたいていレジスタを使うので，カウンタの数だけレジスタの余裕が少なくなります。もし処理で使うレジスタを重複するようであれば何らかの工夫でレジスタの中身を退避する必要があります。ちなみに，このようにループの内外にまた別のループがある状態を「ネスティング」と呼びます。

図2-10

2.16ビットのループカウンタ

ループ・カウンタを16ビットのペア・レジスタにすると，制御できるループ回数が16進で10000H回，10進で65536回になります。これをめいっぱい使うことはまずないでしょうが，ループ・カウンタが8ビットでは困ることというのは多いものです。

ただ，16ビットのカウンタをペア・レジスタで構成した場合の重大な注意点があるのです。それは，

ペア・レジスタの内容をデクリメントしても，フラグは一切変化しない

という点です。念のため，お手もとのZ80命令表の中のフラグの影響について書かれているところで確認してみてください。おそらく「16ビット演算」という項目のところに入れられていると思います。DEC BCとかDEC DEなどという命令は実行後もフラグが変化しないと書

いてあるでしょう。たとえ－1した結果がBC＝0になってもＺフラグは教えてくれないのです。ついでに言うと16ビットのインクリメント命令も同じです。

こうなると，ループ・カウンタのペア・レジスタを－1したあとでＺフラグを見てループの終了を知る，というわけにはいきませんね。ここで16ビットのループ・カウンタを使うときのコツが必要なのです。

ペア・レジスタの内容がゼロかどうかを知る方法には次のようなものがあります。

①ペア・レジスタの一方をＡレジスタに入れ，もう一方と論理和（ＯＲ）をとる。

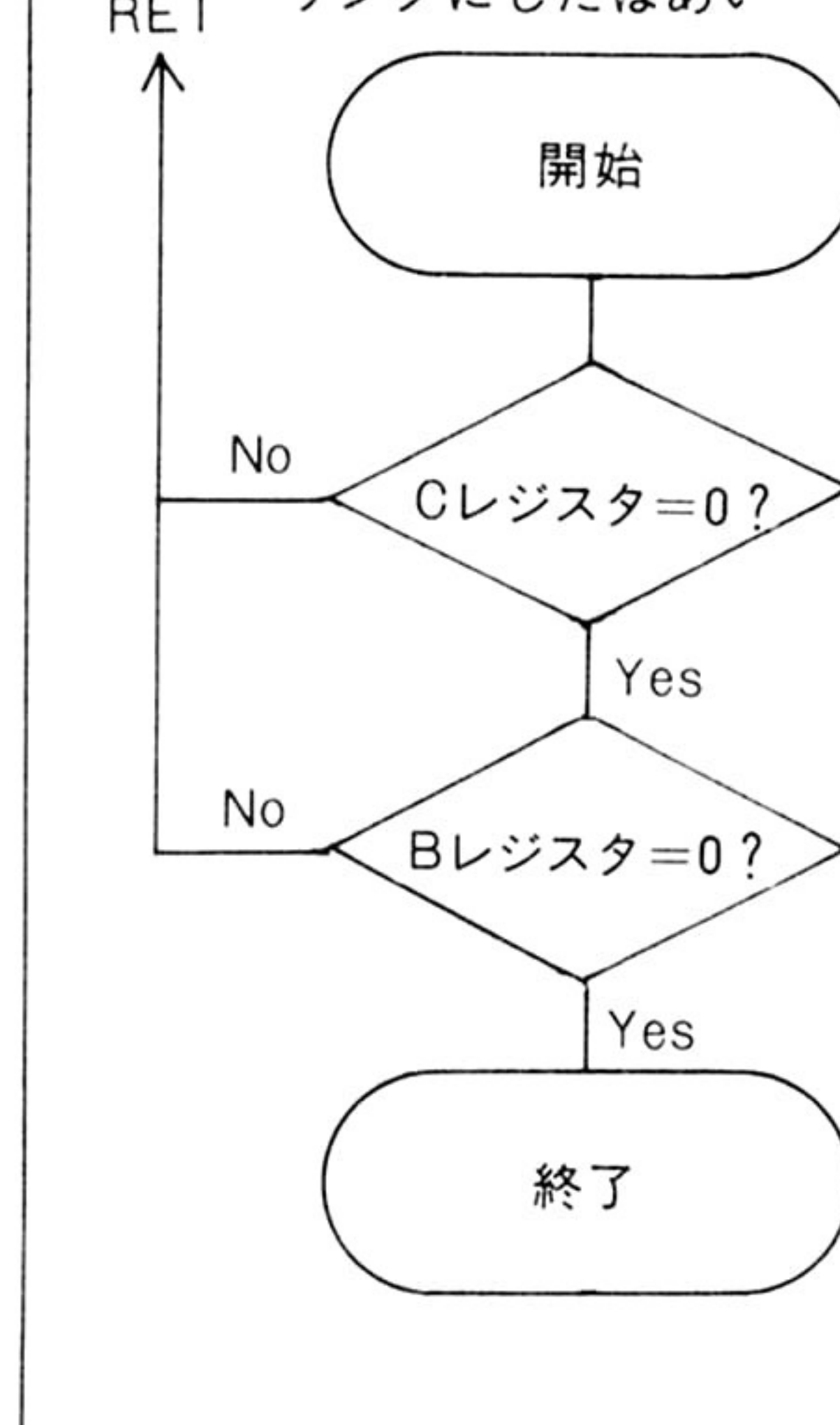
②ペア・レジスタの１つずつをゼロかどうか調べる。

③１を引くことは－1を足すことと考え，ADC命令を使ってカウンタを－1する。ただしこのときはＣフラグにも気を配らねばならない。

①は最もよく使われる方法です。論理演算命令は，Ａレジスタとオペランドとのあいだで論理演算をする命令で，詳しくは別項で取り上げます。そのうち論理和(ＯＲ)とは，対象となる２レジスタの８ビットをすべて比較してゆき，もし１と１とが出会ったら１，１と０なら１，０と０なら０をそれぞれ返す演算です。ですから，もし対象となる２つのレジスタがどちらもゼロであれば結果もすべてゼロになり，Ｚフラグが立つのです。この方法の詳細についてはまたあとで。

図2-11

BCレジスタをループカウンタにしたばあい

②は原始的ともいえる方法です。図2-11のように，まず一方のレジスタ(16ビットの下位のほう)をゼロかどうか調べ，もしゼロならもう一方(上位)を調べにかかるのです。もう一方もゼロなら文句なしにペア・レジスタ＝０ですが，どちらかがゼロでなければ戻ります。

③は少々ウラをかいた方法です。16ビットの演算命令でフラグが変化するひと握りの命令(ADCとSBC)を使います。どちらもHLレジスタと他のペア・レジスタとを２つのオペランドにしてＣフラグとともに加算・減算する命令です。つまり，ループの最後でループ・カウンタを HLレジスタにロードして，DEレ

ジスタ(またはBC，HL，SP)に－1を入れ，さらにCフラグをクリアしてからこれらを足し合わせるのです。すると結果的にHLレジスタの内容は－1されたことになります。SBC命令を使うなら第2オペランドに入れる数値を1にします。

この方法の欠点は，レジスタを多く使うこととCフラグをクリアしなければならないことです。前者はレジスタ内容をどこかに逃がす工夫をし，後者はある種の論理演算をすることで解決できます。この方法の概略を図2-12に示します。

3.DJNZ命令

Z80には，DJNZというニモニックのループ専用命令が用意されています。分類としてはジャンプ命令の一種になり，おおよそ図2-13のような動作をします。つまり，Bレジスタをループ・カウンタにしてループを構成し，カウンタのデクリメントと終了かどうかの判定とを自動的にする便利な命令なのです。DJNZを使おうとするときにはB

図2-12

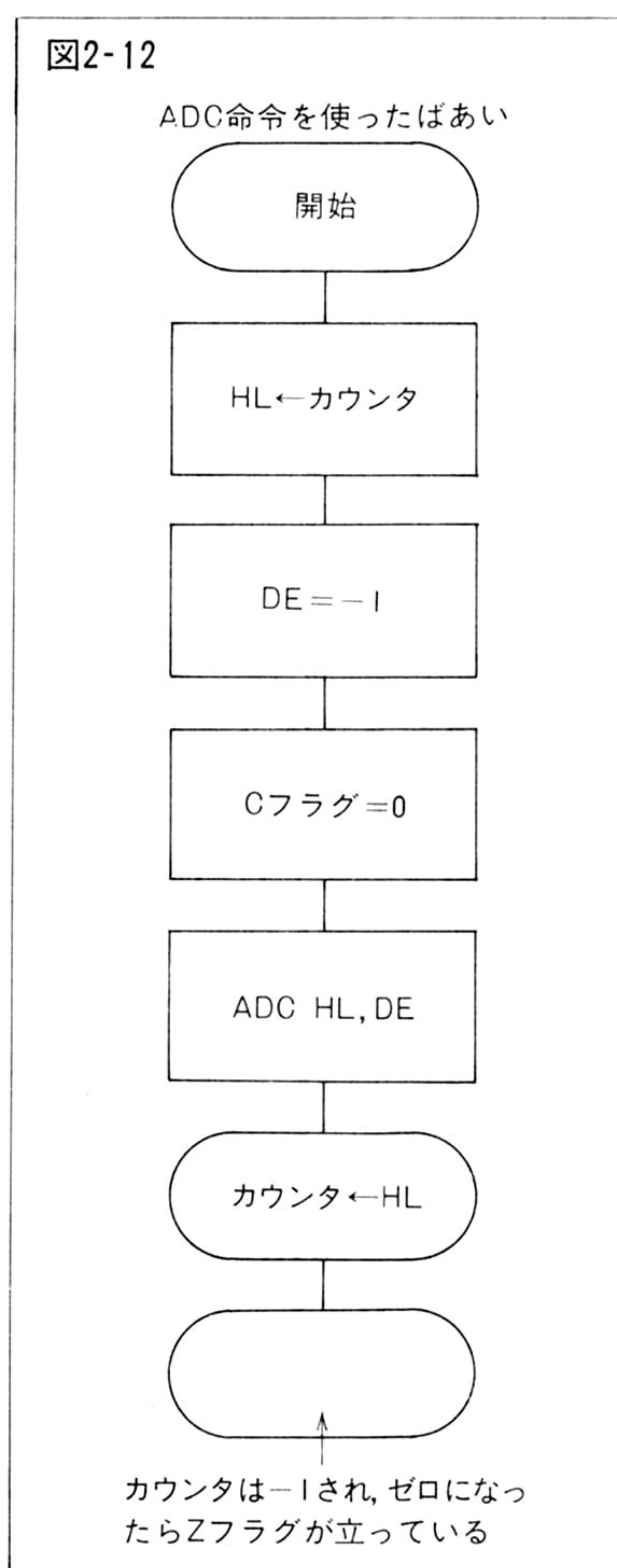
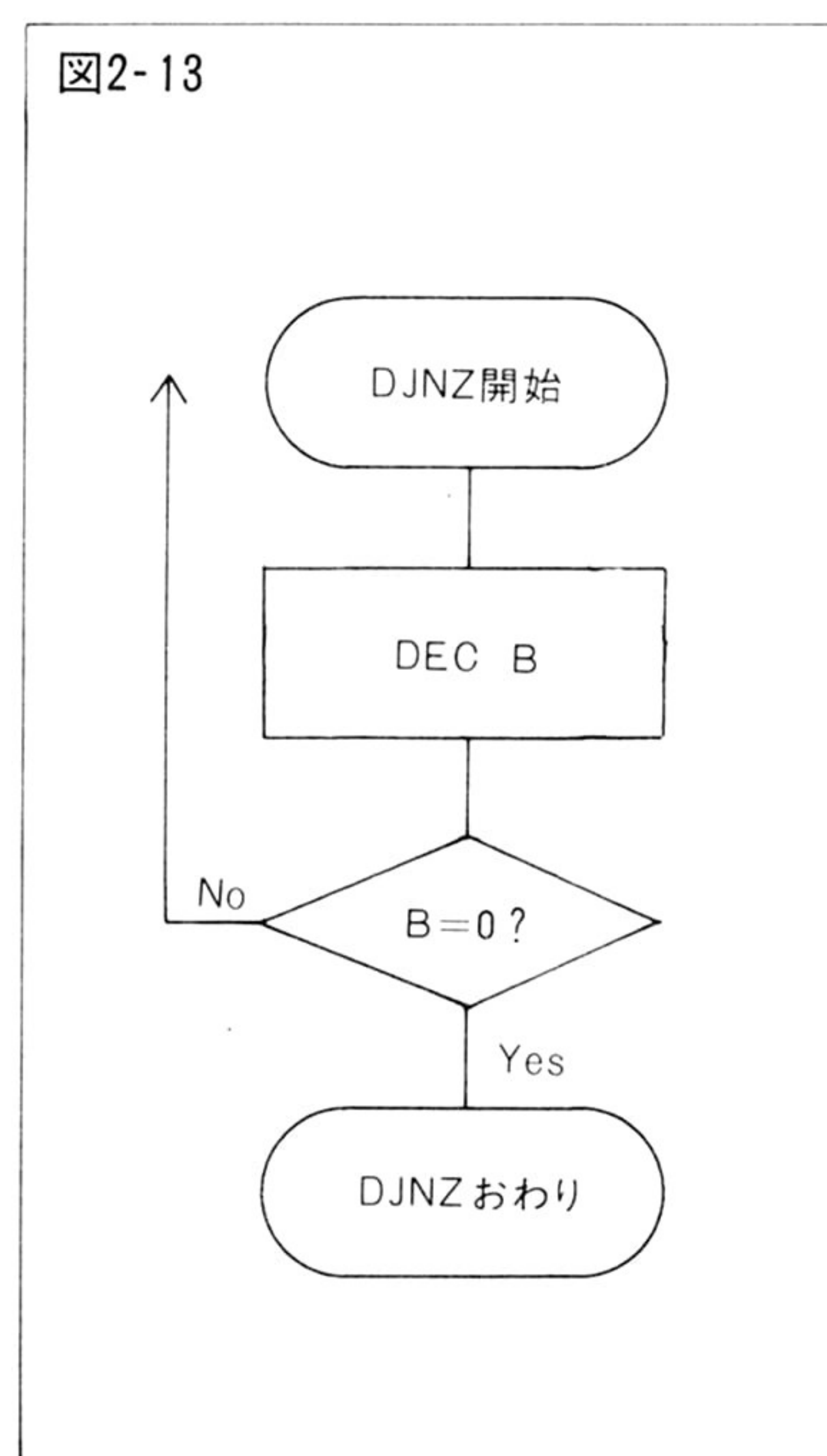



図2-13

レジスタをあらかじめ設定しておく必要があり，また，カウンタはBレジスタ以外には認められていません。ループ中の処理でBレジスタを使わないような，あるいはBレジスタの内容を退避させるような工夫が必要となります。とはいえDECとJR Z命令を組み合わせてループをつくるよりも実行時間が短くすみますし，バイト数も少なくなります。

リスト2-12は，**リスト2-11**をDJNZを使って書きかえたものです。150行でカウンタBがゼロかどうか調べています。ジャンプもするので飛び先のラベルをオペランドにしています。DJNZ命令は相対ジャンプですから，あまり長いループを中間に入れた飛び先は指定できません。**リスト2-12**のほうが1バイト短くすんでいますね。

リスト2-12

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                     ORG      0D400H
110:    D400 2100D3              LD       HL,0D300H
120:    D403 0600                LD       B,0H
130:    D405 75       LOOP:      LD       (HL),L
140:    D406 23                  INC      HL
150:    D407 10FC                DJNZ     LOOP
160:    D409 C9                  RET
```

4.ループの応用例

ここでは，ループを使った例を2～3取り上げてみます。

①時間待ちループ

スピードが魅力でマシン語のプログラムをつくるとしても，場合によってはムダともいえる時間待ちのために工夫せねばならないことがあります。時間待ちのループは，CPUを動かすクロックの周波数と命令のステート数とをにらみながらつくらねば，思うような時間を消費させることはできません。しかも，時間待ちによってレジスタの内容やフラグに影響があってはいけないのです。MSXはどの機種もクロック3.57954MHzですから，1サイクル279n秒です。これがマシン語命令の動作の基礎となる「ステート」の素になります。Z80のマシン語命令表にもし「ステート数」なる欄があれば，その数字に279n秒をかけてその命令を実行するのにかかる時間を知ることができるのです。たとえばLD A，Bという命令は4ステートですね。279n×4＝1.116μ

(秒)となります。

ただ，ここで注意しなければならないのは，MSXの場合,「ウェイト」というものを各命令ごとに1～2ずつ入れているということです。ウェイトをいくつ入れるかは，オペコードのバイト数で決まります。ウェイト1つでステートが1つ増えますから，先のロード命令は

$$279\text{n} \times (4+1) = 1.395\mu\ (\text{秒})$$

の時間を(MSXでは)消費するということになります。

さて，実際に時間待ちループをつくってみましょう。**リスト2-13**を見てください。このループでMSXでは約2m秒かせげます。これを500回くり返せば約1秒の時間待ちになるわけですね。DJNZを使うとネスティングにしにくければ，DECとJRにしてもよいでしょう。ただしこんどはループを抜けたときにフラグが変化してしまうおそれがありますが。

リスト2-13

```
 MSX Self Assembler  Rev 1.0     PAGE    1


100:    D400                     ORG     0D400H
110:    D400 06FF                LD      B,0FFH
120:    D402 03        LOOP:     INC     BC
130:    D403 0B                  DEC     BC
140:    D404 10FC                DJNZ    LOOP
150:    D406 C9                  RET
```

②キャラクタ・サーチ

次はループの中で何か意味のある処理をさせてみます。広大なメモリ空間のうちに指定した1バイトの16進数があるかどうか探し，あればその場所を入れて戻るプログラムを考えてみましょう。

リスト2-14を見てください。最初に130行と140行でサーチするメモリ・エリアのスタート番地，エンド番地をそれぞれSTART，EENという名で定義しています。180で1文字入力し，入力した文字のASCIIコードをこれからサーチする1バイトのデータとします。190行からがループになっています。まず190行でサーチするメモリ・エリアの内容をBレジスタに入れ，200行で目的のデータかどうかチェックします。もしそうだったらZフラグが立つので210行でリターンにジャンプします。そうでなかったら220行でメモリのポインタHLを＋1し，その結果EENアドレスに達してしまったかどうかを230行から270行で調べま

す。230行で(大切な)サーチ・データをCレジスタに逃がしておき，ポインタの上位1バイトが入っているHレジスタをAレジスタに移します。これをEEN上位のDレジスタと比較して，もし同じならZフラグが立ちます。同じでなかったら(Zフラグが立っていなかったら)まだHLがDEにほど遠いということなのでNEXTでAレジスタを元に戻してLOOPにジャンプします。

もし上位が同じなら次に下位を比較します。上位と同じように270行でポインタの下位をAレジスタに移し，Eレジスタと比較します。もしL≧EならCフラグが立たないので290行でリターンに飛びます。もしL<EならポインタがEENに達してないということなのでAレジスタを戻してLOOPにジャンプします。

このプログラムを実行するときは，モニタでGD400, D41A↵」としてください。D41AH番地でリターンしようとするときにブレークさせ

リスト2-14

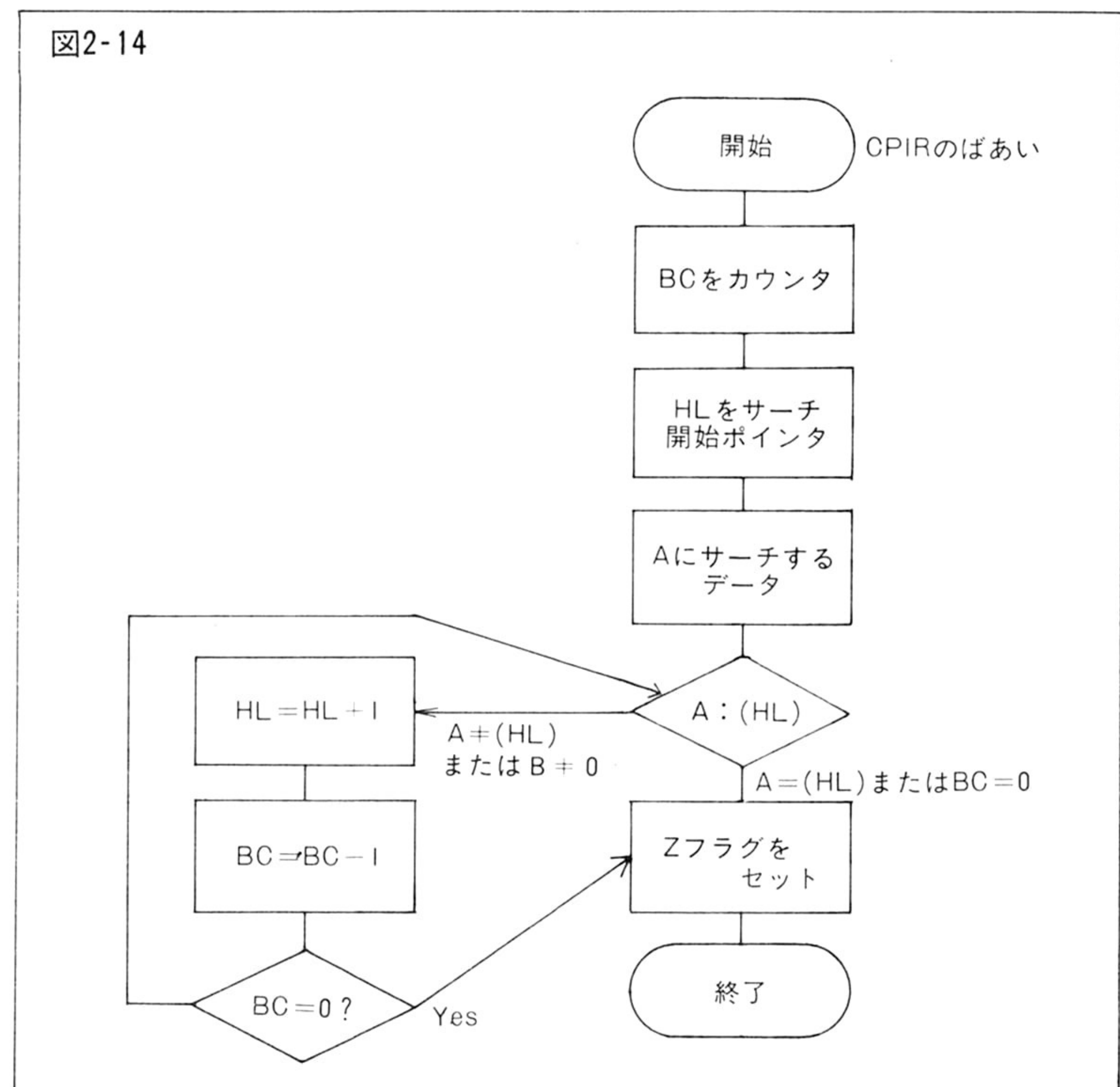
```
 MSX Self Assembler  Rev 1.0     PAGE      1


100:    D400                     ORG       0D400H
110:
120:    009F =         CHGET     EQU       009FH
130:    0000 =         START     EQU       0
140:    2000 =         EEN       EQU       2000H
150:
160:    D400 210000              LD        HL,START
170:    D403 110020              LD        DE,EEN
180:    D406 CD9F00              CALL      CHGET
190:    D409 46        LOOP:     LD        B,(HL)
200:    D40A B8                  CP        B
210:    D40B 280D                JR        Z,RTN
220:    D40D 23                  INC       HL
230:    D40E 4F                  LD        C,A
240:    D40F 7C                  LD        A,H
250:    D410 BA                  CP        D
260:    D411 2004                JR        NZ,NEXT
270:    D413 7D                  LD        A,L
280:    D414 BB                  CP        E
290:    D415 3003                JR        NC,RTN
300:    D417 79        NEXT:     LD        A,C
310:    D418 18EF                JR        LOOP
320:
330:    D41A C9        RTN:      RET
```

て，レジスタの内容をのぞくのです。ブレークしたときAレジスタの内容がEENの下位と同じになっていたら，入力したキーと同じコードがみつからなかったということ，もし入力したキーのコードがAレジスタに入ったままだったら，HLレジスタの内容がそのデータが見つかったアドレスです。もう少しこのプログラムの操作性をよくするためには，16進数による入出力を考えねばなりません。これは今後の課題としましょう。

しかしこの程度のことをするのにこんなに長いプログラムを組む必要があるのかと思いませんか。ほんとうはZ80にはこうした作業にうってつけの命令があるのです。それは「ブロック・サーチ命令」です。先にブロック転送命令を使いましたが，その仲間です。ニモニックはCPIR，CPDR，CPI，CPDの4つで，減算型，加算型，自動型，半自動型がそろっているのもブロック転送と同じです。

ブロック・サーチ命令の動作を図2-14に示します。Aレジスタにはこれからサーチしようとするデータ，HLペア・レジスタにはサーチしようとするメモリ・ブロックのアドレス，BCレジスタにはメモリ・ブロックのバイト数をそれぞれブロック・サーチ命令を実行する前に設定しておかねばなりません。

図2-14

開始
CPIRのばあい
BCをカウンタ
HLをサーチ
開始ポインタ
Aにサーチする
データ
A：(HL)
A≠(HL)
またはB≠0
HL＝HL＋1
BC＝BC－1
BC＝0？
Yes
A＝(HL)またはBC＝0
Zフラグを
セット
終了

リスト2-15は，先の**リスト**2-14をブロック・サーチ命令を使って書きかえたものです。ソースもオブジェクトも目ざましく短くなりました。もっとも，160行の第2オペランドのような手を使うとオブジェクトが理解しにくくなる(逆アセンブルしたときにわかりにくい)ので，このあたりも5～6命令に置きかえることができます。ここで＋1している理由はおわかりですね(たとえば0から10までにはいくつの数字があるかを考えてみてください)。200行でHLを－1しているのは，CPIR命令を抜けるとき(BC＝0またはA＝(HL)でZフラグが立ったとき)には内部でHLを＋1してしまっているからです。つまり，目ざすデータを見つけたときはポインタが次の番地を指しているのです。

このプログラムも，アセンブル後実行するときにはモニタのGコマンドでGD400，D40C⏎とせねばなりません。D40CH番地でブレークしたとき全レジスタを表示するので，AレジスタとHLレジスタの内容に注目してください。Aレジスタがサーチしたデータ，HLレジスタがそのデータを見つけたアドレスです。HLレジスタがプログラム中のEEN＋1になっていたときに限り，Aレジスタと同じデータが見つからなかったか，またはEEN番地に見つけたかのどちらかです。

リスト2-15

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                     ORG      0D400H
110:
120:    009F =         CHGET     EQU      009FH
130:    0000 =         START     EQU      0
140:    2000 =         EEN       EQU      2000H
150:
160:    D400 010120              LD       BC,EEN-START+1
170:    D403 210000              LD       HL,START
180:    D406 CD9F00              CALL     CHGET
190:    D409 EDB1                CPIR
200:    D40B 2B                  DEC      HL
210:    D40C C9                  RET
```

覚えてしまおうマシン語の定石

3 論理演算をマスターする

つぎはコンピュータらしい命令，論理演算命令にうつります。

1.論理演算とは

論理演算という演算がそもそもどのようなものであるかということは抜きにして，ここではその用途と使いかたを述べていきたいと思います。

論理演算はコンピュータの内部がすべて2進によって取りしきられていることを利用しています。しかもこの演算はみなビット単位で行なわれるため，いきおい微細な操作や処理に多く使われるようになります。ただ命令表をながめているだけでは「いつ使うんだろう」と思えてくるような命令ばかりですが，コンピュータ内部で数値を取り扱ったり外部機器を制御したりするときにはなくてはならぬ命令といえます。

基本的は論理演算には4通りあります。すなわちAND(論理積)，OR(論理和)，NOT(否定)，XOR(排他的論理和)です。これらの論理演算が，出会った2ビットをどう処理するかを表にしたのが「真理値表」です。基本6演算について，**表2-2**に真理値表をあげました。真理値表は，XとYの2ビットを入力したとき，それぞれの論理演算によってどのような結果が出力されるかを表にしたものです。

真理値表を見ながら，基本的な3つの論理演算について述べましょう。

表2-2　各論理演算の真理値表

X	Y	NOT X	X AND Y	X ORY	X XOR Y	X IMP Y	X EQV Y
0	0	1	0	0	0	1	1
0	1	1	0	1	1	1	0
1	0	0	0	1	1	0	0
1	1	0	1	1	0	1	1

①AND(論理積)

ANDは，入力された2つのビットがどちらも1のときに限って1を出力します。論理積ということばからして，かけ算に関係がありそうですね。つまり，入力された2ビットをかけ合わせた積が出力となるということです。$0\times0=0$，$0\times1=0$，$1\times1=1$ですから。

ANDは，おもに1バイトのうちの特定のビット群を取り出すときに

使います。ニモニックはANDです。

② OR（論理和）

ORは，入力された2ビットのうち少なくとも1つが1のとき1を，どちらもが0のとき0を出力します。これも論理和ということばどおり，入力された2ビットの値を足したものが出力になっていると考えられます。すなわち，0+0=0，0+1=1，1+1=1です。最後の足し算はおかしいですが，少なくとも「0ではない」のだと理解しておいてください。

ORは，おもに特定のビットを1にするときに使われます。ニモニックはORです。

③ NOT（否定）

NOTは入力が1ビットです。つまり，入力されたビットの値を反転(1→0，0→1)として出力します。

NOTはCPLというニモニックで実現できます。この命令はAレジスタの内容をビットごとにすべて反転するので，結果的にAレジスタの内容の1の補数をとったことになります。1の補数と2の補数の考えかたは，2進数で計算するときの重要な概念ですから,「マシン語入門　基礎編」または類書をごらんください。2の補数だけに関しては本書でものちの6章で解説しています。

2. 論理演算によるビット列操作

さっそく，論理演算を使った処理に取りかかりましょう。まずはビット列操作です。

論理演算で行なうビット列の操作には次のようなものがあります。

①ビットのマスク

②ビットのセットとリセット，はめ込み

③ビットの反転

④ビットの判定

⑤ビットの一致判断

以下，それぞれについてプログラムをあげながら解説していきましょう。

①ビットのマスク

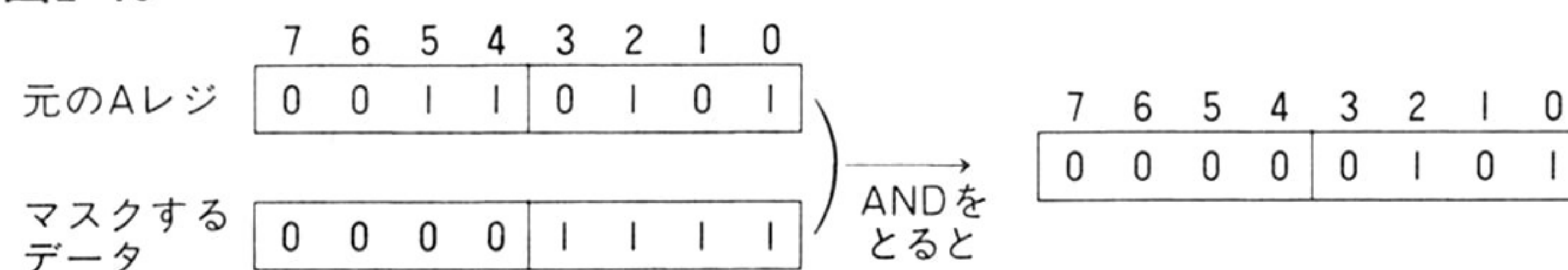
「マスク」とは，あるビット列から特定の数ビットだけを取り出して残りをリセットする（0にする）ことを指すことばです。たとえばAレジスタの下位4ビットだけを取り出したばあい，上位4ビットをすべて0にしてAレジスタに残します。これがマスクです。具体的には図**2-15**のようにします。この例では，マスクされるデータはAレジスタの00110101，すなわち16進数で35Hです。それとマスクするデータ0FHとのANDをとると，元のデータのうち下位4ビットの，しかも1になっているビットだけが残ってあとは0になってしまいます。なぜならANDは1と1とが出会ったとき以外はみな0を出力するからです。もしマスクするデータをF0H（つまり上位4ビットだけ1，あとを0）にすると上位4ビットだけがマスクされて残り，下位はすべて0にできます。

すなわち，この例では35Hを0FHでマスクして05Hを得たことになります。この手法は，特にAレジスタに入っている数値データの16進1桁のみを取り出したいときに使います。

図2-15

	7	6	5	4	3	2	1	0
元のAレジ	0	0	1	1	0	1	0	1
マスクするデータ	0	0	0	0	1	1	1	1

ANDをとると →

7	6	5	4	3	2	1	0
0	0	0	0	0	1	0	1

リスト2-16を見てください。これは，キーボードから読み込んだ文字のASCIIコードの，16進下位1桁がいくつであるかを表示するものです。たとえば“1”のキーが入力されたときはASCIIコードが31Hであるので“1”を，“K”（大文字）が入力されたときはコードが4BHであるので“B”を，それぞれ出力します。

150行で1文字をAレジスタに入力します。160行で下位4ビットにマスクをかけます。160行の実行を終えた時点でAレジスタの内容は上位4ビットが0000，下位4ビットだけが残されています。ですから，このときのAレジスタの内容は00Hから0FHのあいだのどれかです。そこで，170行で0AHと比較します。なぜ0AHかというと，“0”から“F”までの文字コードのうち，“9”と“A”との間にズレがあるからです。ASCIIコード表を見ていただくとわかりますが，“0”～“9”までが30H～39H，“A”～“F”までが41H～46Hとなっていますね。これがもし続いているものなら170行のような場合分けをしないでもよいのです。

もしAレジスタの内容が00H～09Hまでなら170行でCフラグが立ちます(Aレジ＜0AH)。Cフラグが立っていれば180行でNINEにジャンプし，Aレジスタの内容に30Hを加えます。00H～09Hまでなら30Hを加えるとちょうど“0”～“9”までのASCIIコードになりますね。もし170行でCフラグが立たなければ0AH以上ということですから，単に30Hを加えても“A”～“F”にはなりません。そこで190行で余分に7を加えます。200行でさらに30Hを加えれば結局0AH～0FHには37Hを加えたことになり，“A”～“F”までのASCIIコードになりますね。こうして求まったコードを210行で画面に表示してリターンです。

リスト2-16

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                     ORG      0D400H
110:                 ;
120:    009F =       CHGET       EQU      009FH
130:    00A2 =       CHPUT       EQU      00A2H
140:                 ;
150:    D400 CD9F00              CALL     CHGET
160:    D403 E60F                AND      0FH
170:    D405 FE0A                CP       0AH
180:    D407 3802                JR       C,NINE
190:    D409 C607                ADD      A,7H
200:    D40B C630    NINE:       ADD      A,30H
210:    D40D CDA200              CALL     CHPUT
220:    D410 C9                  RET
```

②ビットのセットとリセット，はめ込み

Z80にはある特定の1ビットをセットしたりリセットしたりする命令がありますが，複数ビットをまとめてセット，リセットしたいときは論理演算を使ったほうが速くて便利です。また，ある1バイトの上位4ビットに他の1バイトの上位4ビットの情報をそのままはめ込みたいときにも論理演算が使えます。

ビットのリセットにはANDを使います。マスクもその一種で，リセットしたいビットを0に，残したいビットを1にしておいてから対象の1バイトとANDをとると，目的のビットだけがリセットできます。真理値表のANDの項を見てわかるように，一方が0であれば他方は1であっても0であっても結果は0であることを利用しています。例をP60の**図2-16**にあげておきました。

ビットのセットにはORを使います。リセットと同じように，セットしたいビットを1，リセットしたいビットを0にしたパターンをつくって対象の1バイトとORをとるのです。すると**図2-17**のように目的のビットを1にできます。

ビットのセット，リセットどちらの場合もプログラムによる例はあげませんでしたが，それぞれ図中にある方法でビット操作をした後，1章の2の**リスト1-13**と組み合わせてビット内容を確認するとよいでしょう。

ビットのはめ込みにもORを使います。ビットのはめ込みというのがわかりにくい人は**図2-18**を見てください。この例ではあらかじめ2つのビット・パターンを「上位4ビットだけマスク」「下位4ビットだけマスク」にしておいてORをとり，結果として元の上位・下位各4ビットずつを組み合わせています。この例のような4ビット単位に限らず，はめ込もうとする相手のビットをあらかじめクリアしておくことによって任意のビット数・位置ではめ込みが実現できます。

ビットのはめ込みは，上・下位4ビットを単位とするBCD表現の数を組み合わせるときなどに使います。

図2-16

偶数ビット(0、2、4、6)を
リセットする

	1	1	1	0	1	1	0	0	Aレジ＝ECH
AND	1	0	1	0	1	0	1	0	パターン＝AAH
	1	0	1	0	1	0	0	0	結果＝A8H
		☆				☆			☆がリセットされた

A＝0ECHとしておいて
AND　0AAH
↓
A＝0A8Hとなる

図2-17

奇数ビット(1、3、5、7)を
セットする

	0	0	0	1	0	0	1	1	Aレジ＝13H
OR	1	0	1	0	1	0	1	0	パターン＝AAH
	1	0	1	1	1	0	1	1	結果＝BBH
	☆		☆		☆				☆がセットされた

A＝13Hとしておいて
OR　0AAH
↓
A＝0BBHとなる

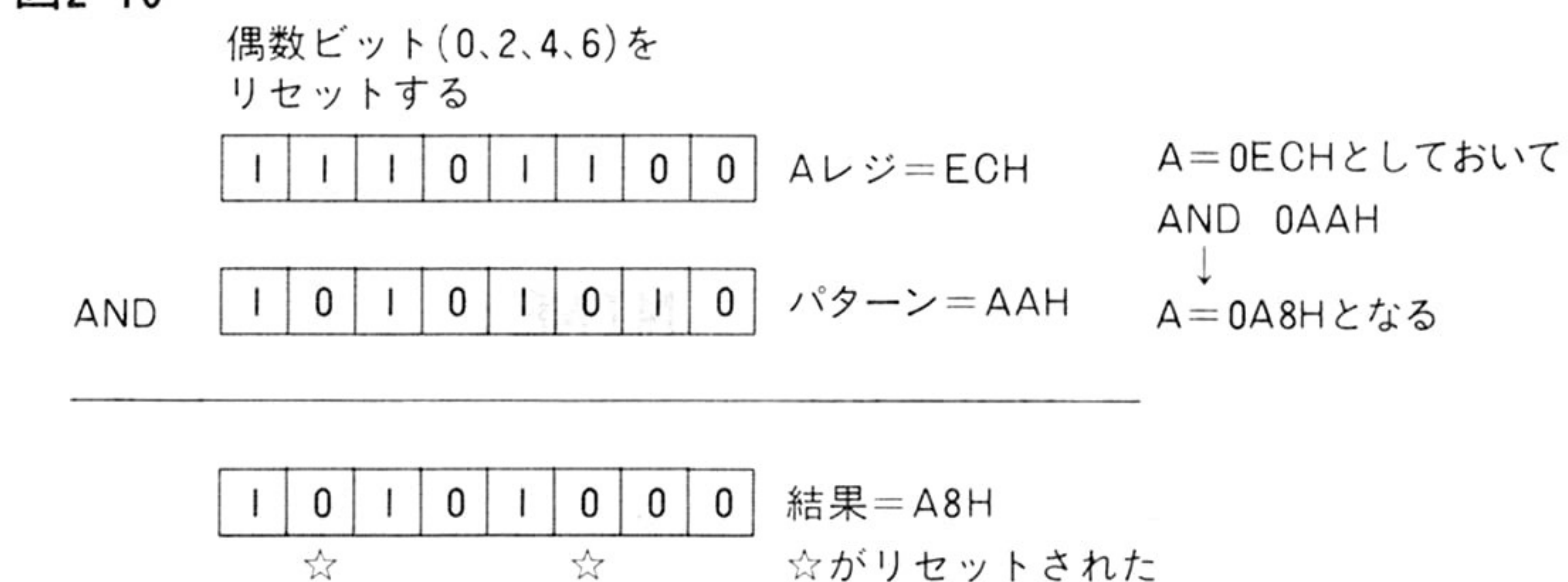
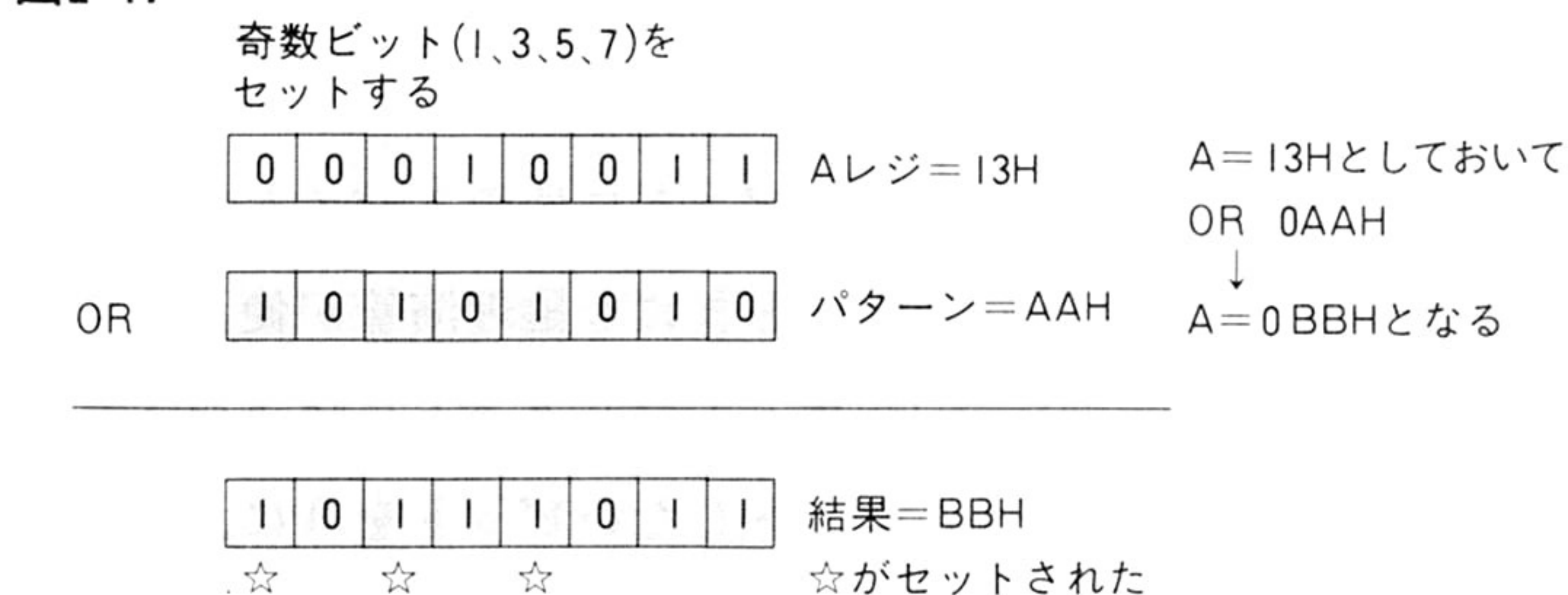
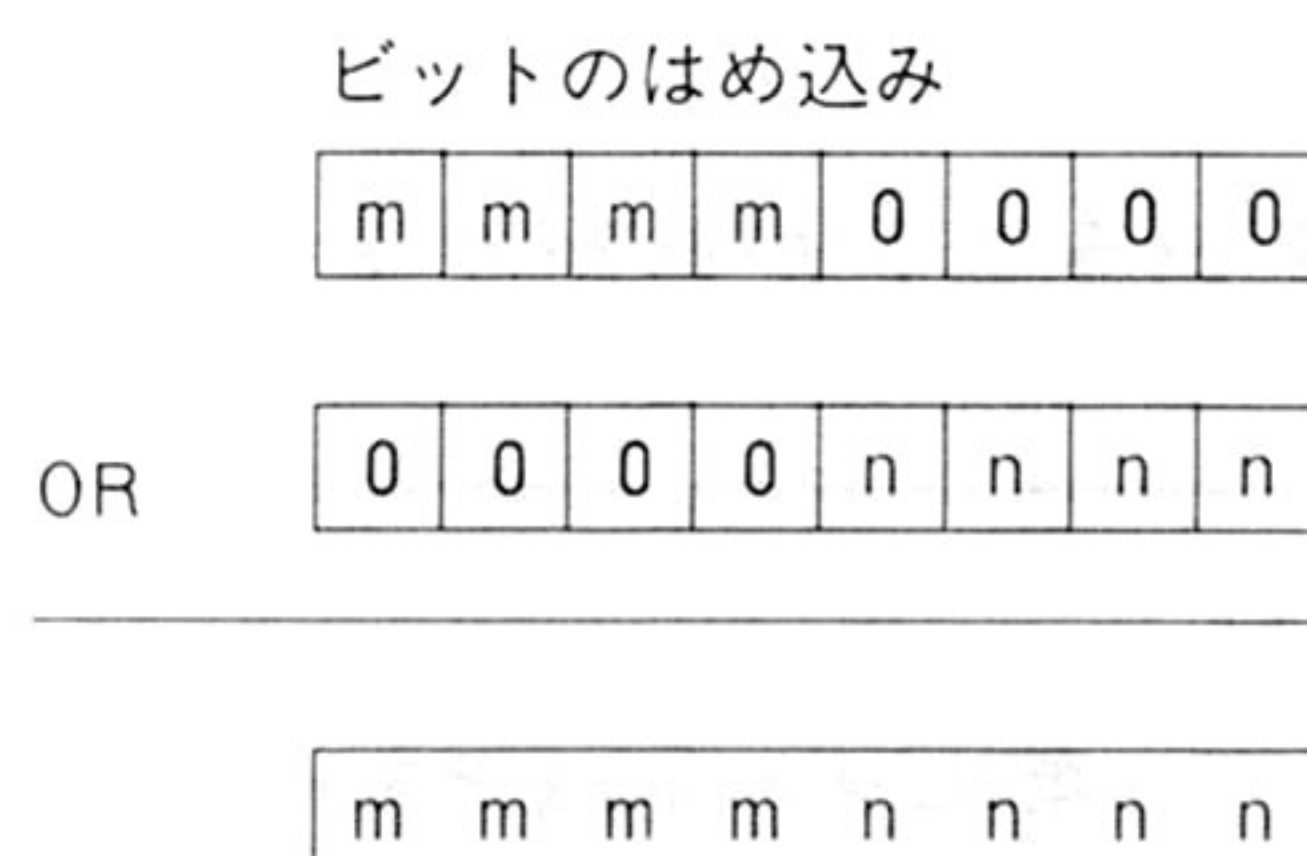
図2-18

ビットのはめ込み

m	m	m	m	0	0	0	0

OR

0	0	0	0	n	n	n	n

m m m m n n n n

③ビットの反転

任意のビット列を反転(0→1, 1→0)にしたいときはXORを使います。XORは入力されたビットが同値のとき0を,異なるとき1を出力する演算ですから,反転したいビットを1に,残りを0にしたパターンをつくって対象の1バイトとXORをとれば反転できるわけです。**図2-19**を見てください。この場合は全ビットを反転するため,XORをとるパターンを全ビット1(=0FFH)にしています。もし7, 5, 3, 1ビットを反転したいのなら10101010(16進で0AAH)とXORをとればよいのです。パターンのうち,0になっているビットは結果に何の影響も与えません。

図2-19

全ビットを反転する

	1	0	1	1	1	0	1	1	Aレジスタ=BBH
XOR	1	1	1	1	1	1	1	1	パターン=FFH
	0	1	0	0	0	1	0	0	結果=44H

A=0BBHとしておいて
XOR 0FFH
↓
A=44H

④ビットの判定(テスト)

対象とする1バイトのうち目的のビットが1であるか0であるかを判定するにも論理演算が使えます。もし目的が1ビットだけならZ80のBIT命令でテストできますが,複数ビットを同時に判定したいときにはBIT命令は不向きです。

ビットのテストにはANDを使います。テストしたいビットを1,他を0にしたパターンをつくり,対象の1バイトとANDをとると,テストしたいビットはそのまま残り,他のビットはすべて0になった結果が残ります。そこでこの結果の値が00Hであるかどうかを調べればよいのです。ゼロを調べるのはZフラグですね。もし結果がゼロであれば(Zフラグが立っていれば)テストしたビットはゼロだったとい

うことですし，ゼロでなければテストしたビットが1だったということです。図2-20に例を示します。この場合，目的の第2ビットは1だったということがわかります。

図2-20

第2ビット(b_2)をテストする

	1	1	0	0	1	1	0	0	
	1	1	0	0	1	1	0	0	Aレジスタ＝CCH
AND	0	0	0	0	0	1	0	0	パターン＝04H
	0	0	0	0	0	1	0	0	結果＝04H

Zフラグ＝リセット→第2ビットは1だった

⑤ビットの一致判断

　異なるバイトの対応するビットが一致しているかどうかを判断することも論理演算でできます。つまり，たとえばある1バイトの第2～4ビットと別の1バイトの第2～4ビットとが完全に一致しているかどうかの判断ができるわけです。

　この場合，一致をみる方法には2通りあります。1つはマスクを使う方法で，たとえばM番地の第2～4ビットとN番地のそれとを比較したいときには

```
LD A,(M)
AND 00011100B
```

と一方のビット列をマスクして目的のビットだけを残し，

```
LD B,A
```

と退避しておいて，

```
LD A,(N)
AND 00011100B
```

ともう一方のビットもマスクしてから

```
CP B
```

とすれば，一致ならZフラグが立ちます。

　もうひとつの方法はXORを使う方法です。すなわち

```
LD A,(M)
LD HL,N
```

としておいて

```
XOR (HL)
```

とすると，(M)と(N)とで一致したビット列の部分だけAレジスタに

は0が残ります。そこで一致を見たいところを残して

AND 00011100B

とマスクをかけると，もし目的のビットがXORによって0になっていれば(すなわち一致していれば) ANDによってZフラグが立ち，そうでなければ立ちません。この2つの方法を図示したのが図2-21です。

これら2つの方法のうち，どちらを採用するかは自由です。ただXORを使う方法のほうがソースもオブジェクトも短くてすみます。A以外のレジスタへの影響はXORを使う方法のほうが大きい(HLを使う)です。

図2-21

○最初にマスクをかけてCPする方法

(M)＝ →マスクして 000 00

(N)＝ →マスクして 000 00

CPして斜線部が
一致すればZフラグが立つ

○XORとANDを組み合わせる方法

432

(M)＝

(N)＝

XORをとる

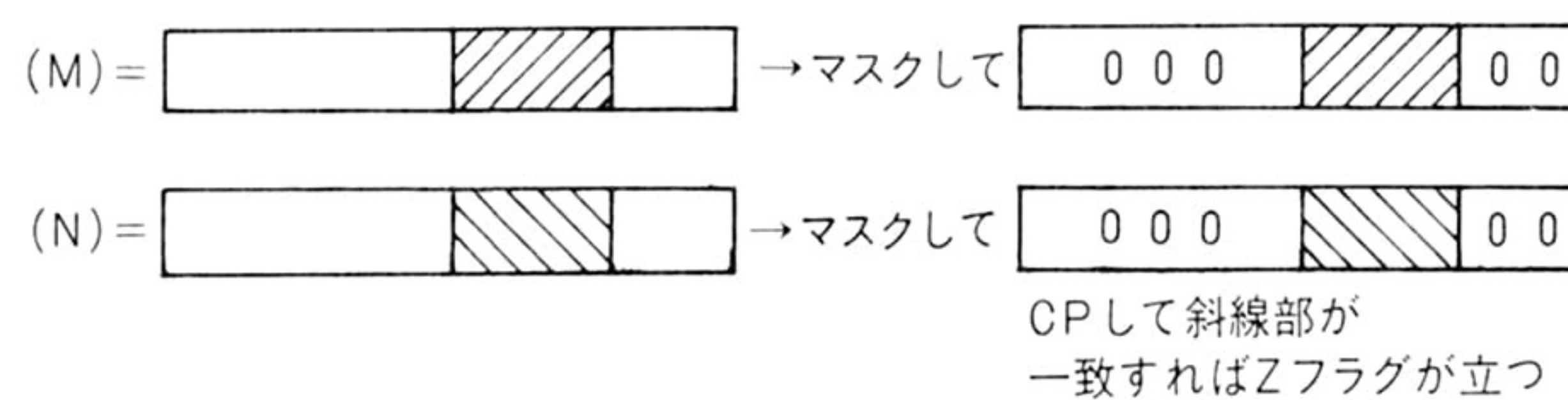
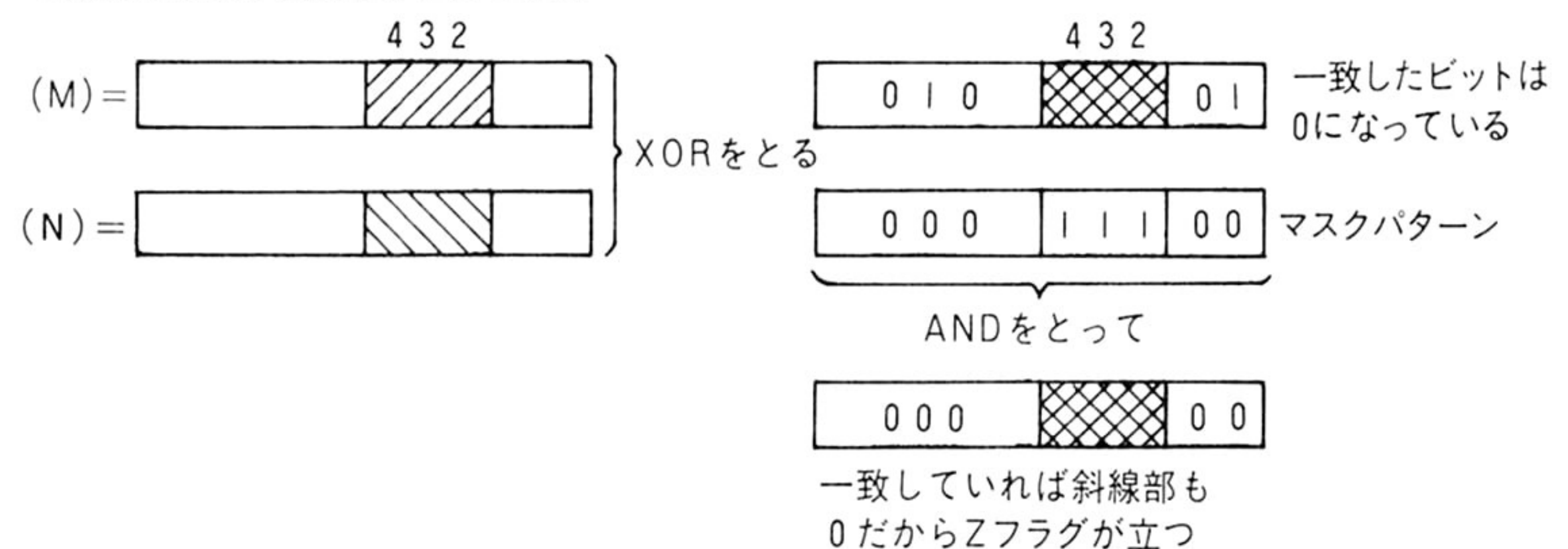
432

010 01 一致したビットは0になっている

000 111 00 マスクパターン

ANDをとって

000 00

一致していれば斜線部も
0だからZフラグが立つ

3. フラグへの影響

つぎに，論理演算命令がフラグに及ぼす影響についてみてみましょう。

基本的な4つの論理演算命令のうち3つ(AND, OR, XOR)はどれもフラグの影響がほぼ同じです。すなわち，S, Zフラグは実行結果によって変化し，C, Nフラグはリセットされ，P/VフラグはPになります。わずかにHフラグだけはANDでセット，他でリセットになります。もう1つの論理演算NOTにあたるCPL命令は，HとNフラグがセットされるほかはフラグに影響しません。

これらのことをふまえたうえで，各主要フラグ別に解説していきましょう。

① Zフラグ

CPL命令を除き，論理演算をする命令はみなZフラグに影響を及ぼ

します。すなわち，論理演算をした結果が0になれば(Aレジスタ＝0) Zフラグが立ち，0でなければ立ちません。結果が0になるということは，各々の演算において被演算数間の関係が次のようであったことを表わしています。

○AND命令→被演算数の少なくとも一方が0であった。

○OR命令→被演算数の両方とも0であった。

○XOR命令→被演算数が同じ値だった。

このうち，もっともよく使われるのがOR実行後のZフラグです。たとえばペア・レジスタの内容を－1した後に0になったかどうかを知りたいばあい，DECによるZフラグの変化は期待できません。そこでORを使います。P33の**リスト2-1**を見てください。このように，調べたいペア・レジスタの一方をAレジスタにロードし，もう一方のレジスタとORをとるのです。こうすれば，ペア・レジスタが0000HであればORによってZフラグが立つのです。このばあい，元あったAレジスタの内容は破壊されてしまうので壊したくなければ工夫してください。

②Cフラグ

Cフラグもまた Zフラグと同じようにAND, OR, XOR命令で影響を受けます。ただしCフラグのばあいは，結果の値によってフラグの状態が変化するというのではなくて，これらの命令を実行すると必ずクリアされます。このこともまた多くの用途があります。

というのは，16ビットの加減算命令を使うときにCフラグをリセットしなければならないことがあるからです。Z80の16ビット加算命令にはADDとADC命令があり，Cフラグを計算に含めたいとき(32ビット＋32ビットの計算で下位16ビットの桁上がりをも加えたいとき)にはADC命令を，そうでないときにはCフラグを加えないADD命令を，それぞればあいによって選ぶことができます。

ところが16ビットの減算命令にはSBCしかなく，SUB命令はありません。この理由は明らかでありませんが，理由はともかくCフラグまで減算したくないときは何らかの方法でCフラグをリセットする必要があるのです。

しかもCフラグだけを操作する命令にはSCF (Cフラグをセットする)とCCF(Cフラグを反転する)の2通りしかなく，Cフラグをリセットする命令はありません。もちろんSCFとCCFとをこの順で実行すれば必ずCフラグがリセットできるのですが，このような方法をとるこ

とはまずありません。

そこで論理演算によるCフラグのリセットが活躍します。論理演算は1命令1バイト(レジスタをオペランドにしたとき)で，さきほどのSCF+CCF(2バイト)よりもスマートです。

Cフラグだけをリセットして，Aも含めた全レジスタの内容を変化させないようにするには，OR AまたはAND Aを使います。このようにオペランドをAレジスタにすればAレジスタどうしを論理演算したことになります。ANDやORは被演算数が2つとも同じであると結果も同じになるので，結果的にAレジスタに残る数に変化がありません。その点Cフラグをクリアするときにはうってつけの命令といえましょう。XOR Aは同じようにCフラグをクリアしますが同時にAレジスタの内容をもクリアしてしまうので，そうなってもよいときでなければ使えません。

③Sフラグ

Sフラグはサイン・フラグとも呼び，演算した数が2の補数で表現されているとみたときの符号(最上位の第7ビット)を保存しているフラグです。いまAレジスタの内容になっている数値が2の補数による負数(80H～FFH)であるかどうかを調べるには，わざわざその値そのものが80H～FFHであるかどうかを確認しなくても論理演算によるSフラグの変化を調べればよいのです。

AND A，OR A，XOR Aとも，Aレジスタの内容が80H～FFHであればSフラグをセットします。しかもAND, ORはAレジスタの内容を変化させないので好都合です。XORはAレジスタをクリアさせてしまうのでうまくありません。

4. 論理演算の応用

①レジスタのクリア

あるレジスタの内容をクリア(0にする)したいとき，もっともオーソドックスな方法はやはり0をロードすることでしょう。しかしどのレジスタであっても0をロードする命令はオブジェクトが2バイトになり，しかもMSXで8ステートと時間も多くかかります。

Aレジスタをクリアするなら，LD A,0とするよりもXOR Aとしたほうが速くて短くすみます。ただしもしフラグに影響を及ぼしたくなければロード命令を使うほかありません。

A以外のレジスタをゼロにするには，XOR Aほどスマートな方法がありません。もしすでにAレジスタをクリアしているのなら目的のレジスタに次々とAレジスタの内容(0)をロードしていけばよいので

すが，そうそう都合のよいことばかりでもないでしょう。この方法は，プログラム最初の初期設定に使えます。

メモリをクリアするのも，Aレジスタをクリアしておいてから LDIR や DJNZ 命令を使って次々とメモリ・ブロックにAレジスタをロードしていく方法があります。

②2の補数への応用

16ビットの数どうしの減算をするのに，Z80 では SBC 命令しかないためあらかじめCフラグをクリアしなければならないことは先に説明しました。この方法はフラグ以外には何の影響も与えないので大変スマートでもっともよく使われます。

しかし，引き算を負数の足し算とみて計算することもできます。ある2進数を負数にするには，2の補数表現を使いますね。2進数を2の補数表現にすることは，全ビットを反転して +1 すればよいので，2の補数にしたい数が入ったレジスタの内容に NOT をかけてから +1 します。

つまり，もし10進数の120（2進数の01111000）を2の補数表現による −120 にしたいなら以下のようにします。

```
01111000 …… 元の数。120。
   ↓
10000111 …… 反転させた(CPL命令)。
   ↓
10001000 …… +1 した。−120。
```

こうしてできた10001000は2の補数としてみると −120 ということになります。

Z80 にはこれを1命令でする NEG という命令があります。NEG は Aレジスタの内容を2の補数にする命令で，CPL とは少しちがっています。

NEG 命令で引き算の一方の数を2の補数表現による負数にし，その後加えることによって16ビットの減算になるわけです。いまHLレジスタに入っている数からDEレジスタの数（12000とする）を引く計算をしたいとすると，**図2-22**のような手順をふめばよいことになります。

図2-22

① DEを2の補数にする。
1) A←E
2) NEG
3) E←A
4) A←D
5) CPL
6) D←A

② HLとDEを加える。

覚えてしまおうマシン語の定石

4 スタックについて

本章最後のテーマは,プログラミング最後のとりで「スタック」です。

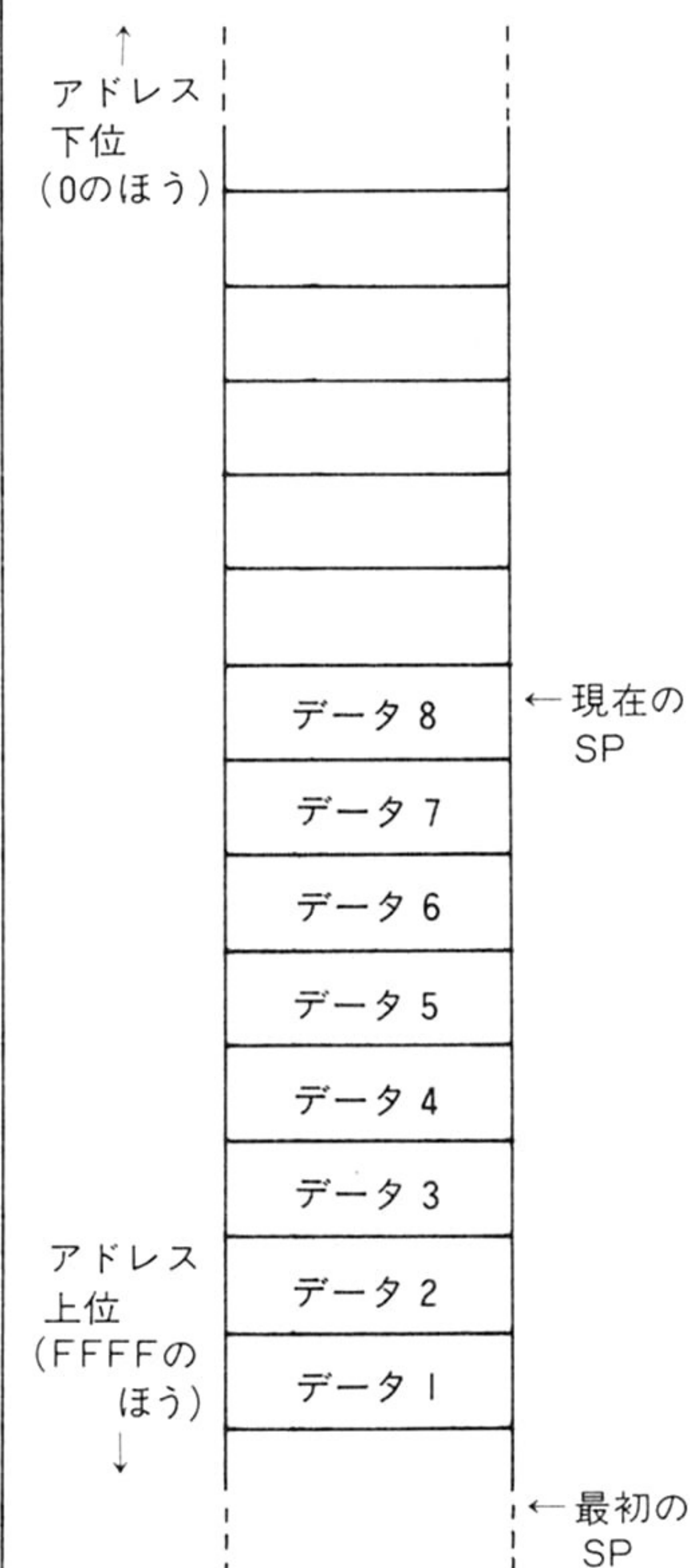
1. スタックとは

メモリの一部を特別な目的のために使っているのがスタック領域です。スタックはRAMの中の指定した番地からアドレスが少ない方向(下位という)に成長していきます。スタックのデータはFILO(First In, Last Out)という方式で入出力されます。これは「最初にスタックに入れたものが最後に取り出せる」という意味です。逆に言えば最後に入れたものを最初に出さねばならないということです。

メイン・メモリ中のどこをスタックとして使うか,あるいは使っているかという情報は,CPUの中のスタック・ポインタ(SP)というポインタにしまわれています。SPが指している番地は,これまでに入れたデータの頂上の番地です。スタックのようすを模式的に示したのが**図2-23**です。この図ではスタックにすでに8個(8バイト)のデータが格納されています。現在のSPはデータ8の格納された番地を指しています。この状態でさらに2バイトのデータをスタックに入れると,データ8のある場所の上(アドレス下位)にデータ9,さらにその上にデータ10を格納します。

データをスタックから取り出すときはその逆です。**図2-23**の状態から2バイトのデータを取り出すなら,まずデータ8,つぎにデータ7の順で取り出します。**図2-23**の状態から突然データ4や3を取り出すことはできません。これがFILOの考えかたです。

スタックのエリアをメイン・メモリ内のどこから始めるかは,プログ

図2-23

ラムの最初で指定せねばなりません。**図2-23**では「最初のSP」と書かれているところが，スタック・エリアの開始番地です。

Z80にはSPを操作する命令がありますが，いたずらにSPをいじると暴走や原因不明のバグに悩まされたりするので十分注意しましょう。また，**図2-23**でアドレス下位のほうにどんどんスタック・エリアが伸びていくとそのうちユーザープログラム自身と重なってしまうような気がしますが，普通の使いかたであればそれほど大量のスタックを使うことはありません。スタックの初期設定はメイン・メモリのできるだけ上位のほうにしましょう。

この項ではスタックを使うZ80マシン語命令の双璧であるCALL命令とPUSH命令のグループについて考えてみたいと思います。

2. PUSH, POP 命令

① PUSH, POPとは

スタックの操作をもっとも気軽にできるのがPUSHとPOPの両命令です。これまで，レジスタの内容を逃がさなければならないときはワーク・エリアにセーブしたり他の空いているレジスタにロードしたりしていました。PUSH, POP命令があれば，みなスタックに退避させることができるのです。一時的な退避のためなら，むしろスタックを使うほうがノーマルともいえます。

例としてカウンタの退避をしてみましょう。Bレジスタをカウンタとし，MSXの1文字出力のBIOSを使って指定の文字数だけ画面に文字を表示する，というルーチンをつくってみます。これならできますね。**リスト2-17**です。表示する文字は9～0の数字で,9から始めてゼロまで表示したらスペースを1つあけ,再び9～0まで表示ということを10回繰り返すのです。

Bレジスタがカウンタですから，ループの制御にはDJNZ命令を使っています。しかも，9～0までの数字をプリントするのに1つ，それを10回くり返すのにもう1つと，計2つのDJNZ命令があるのです。

140行のCRLFは280行からに定義されたサブルーチンです。これはASCIIコード 0DHと0AHをプリントします。この2つのコードをこの順にプリントすることによって改行できます。決まり文句と思ってください。

L00P1が外側，L00P2が内側です。内側は170行からで,180行のAレジスタの2FHにカウンタ10～1を足す(190行)ことによってAレジスタに39H～30Hが残ります。これらは“9”～“0”までのASCIIコードに対応しているため，200行で“9”～“0”までの文字が表示されます。

210行で10文字プリントしたかどうか調べ，10文字終わっていたら220～230行でスペース(ASCIIコード20H)をプリントします。

外側のループは内側のループを10回繰り返すだけです。150行で最初のカウンタを設定してから，いったん160行でスタックに逃がします。そのあと170～210行の内側ループを実行し，スペース1文字をプリントしてから240行でスタックに逃がした外側のカウンタを復帰し，250行で10回まわったことを確認します。

この例ではネスティングした2つのループカウンタに同じBレジスタを使ったため，外側のカウンタをどこかに退避させなければなりませんでした。退避先はスタックでなくともよく，この場合空いているD，E，H，Lなどのレジスタでもよかったのです。ただ，PUSH，POPをしたほうがプログラムも見やすくなります。

リスト2-17

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                     ORG      0D400H
110:                 ;
120:    00A2 =       CHPUT       EQU      00A2H
130:                 ;
140:    D400 CD19D4              CALL     CRLF
150:    D403 060A                LD       B,10
160:    D405 C5      LOOP1:      PUSH     BC
170:    D406 060A                LD       B,10
180:    D408 3E2F    LOOP2:      LD       A,2FH
190:    D40A 80                  ADD      A,B
200:    D40B CDA200              CALL     CHPUT
210:    D40E 10F8                DJNZ     LOOP2
220:    D410 3E20                LD       A,' '
230:    D412 CDA200              CALL     CHPUT
240:    D415 C1                  POP      BC
250:    D416 10ED                DJNZ     LOOP1
260:    D418 C9                  RET
270:                 ;
280:    D419 3E0D    CRLF:       LD       A,0DH
290:    D41B CDA200              CALL     CHPUT
300:    D41E 3E0A                LD       A,0AH
310:    D420 CDA200              CALL     CHPUT
320:    D423 C9                  RET
```

②レジスタの内容の交換

PUSH，POP 命令はたいてい対になって使われます。PUSH したまま，あるいは POP したままでいるとスタックのレベル(どこまで Push したか)のつじつまが合わなくなります。

ただ，DE レジスタの内容を PUSH したら必ず DE に POP せねばならないということはありません。元に戻らなくてもかまわないなら他の使いかたもできるのです。

たとえばレジスタの内容交換です。BC レジスタ ↔ HLレジスタを通常の方法でするなら

```
LD A, B ┐
LD B, H ├ B↔H
LD H, A ┘
LD A, C ┐
LD C, L ├ C↔L
LD L, A ┘
```

と 6 命令を要します。ところがスタック経由にすると

```
PUSH BC
PUSH HL
POP BC
POP HL
```

と 4 命令ですみます。単に LD BC，HL(このような命令はない)という動作をさせたいだけなら

```
PUSH HL
POP BC
```

でよいのです。

ただ，DE と HL との間の内容交換などには独立した命令があるので，そっちを使ったほうがスピードもバイト数もかせげます。逆に BC と DE 間やインデックス・レジスタの関係などには 16 ビットどうしのロードや交換命令がないので，スタックを経由する方法を使うほうがよいでしょう。

③1バイトのPUSH, POP

PUSH命令，POP命令とも16ビットまとめてスタックに出し入れするので，たとえばPUSH AFのあと何かのサブルーチンを実行し，戻ってきたのちPOP AFをすると，PUSHしたときのF(フラグ・レジスタ)がPOPで復帰してしまい，困ります。サブルーチンの前後でフラグを保存しておく必要がもしあるなら，このことは歓迎すべきなのですが，サブルーチンを実行した結果をフラグに入れて戻ってくるような（たとえば文字列を検索して同じものがあればZフラグを立てて戻るサブルーチンなど）場合は，戻ったとたんPOP AFとされたら困るのです。

そこで1つのレジスタだけをスタックにしまう方法を考えたのが**リスト2-18**です。200行と210行のPOP BCとLD A, BでAレジスタだけを復帰しています。この方法だとBCレジスタが壊されてしまいますので，DEでもHLでも空いたペア・レジスタを使ってください。もしどのペア・レジスタも破壊不可能な内容なら，最初からPUSH AFとせずにAレジスタだけメモリかどこかに退避させておけばよいでしょう。

リスト2-18

```
100   ORG    0D400H
110   PUSH   AF
        :
200   POP    BC
210   LD     A, B
        :
```

また，レジスタではなくメモリの内容をスタックにPUSH, POPすることも工夫すればできます。HLを使いますがその内容は破壊されません。すなわち，

```
PUSH HL
LD HL, (9000H) ; 9000HはPushしたいメモリ
EX (SP), HL
```

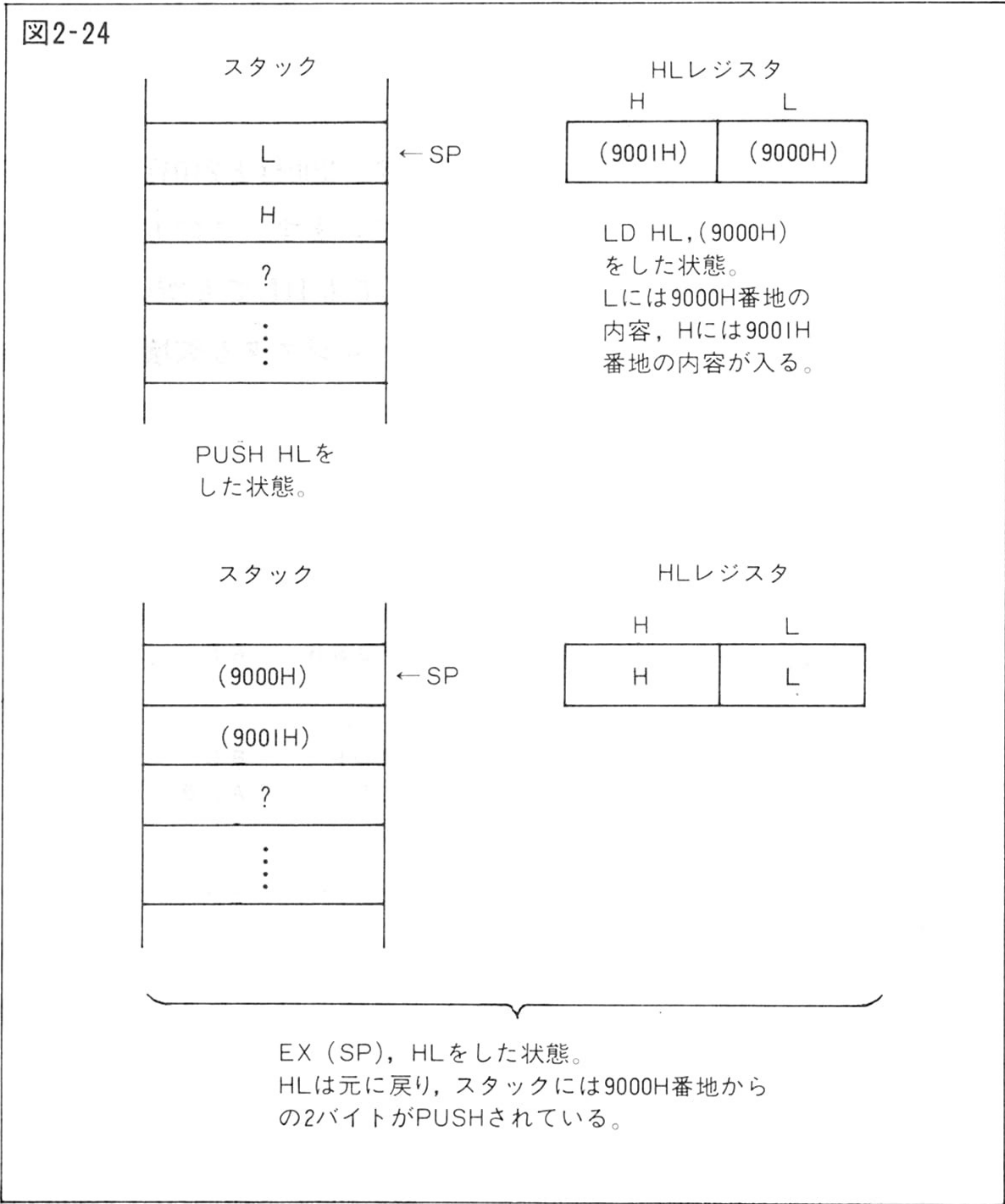
という方法です。ここで出たEX (SP), HLは，そのときのスタック・トップから2バイトの内容とHLレジスタの内容とを交換する命令です。つまり，いったんHLをPUSHしてからこのEXを実行すれば，そのときのスタック・トップの2バイト(PUSHされたHL)と新しいHL(この例では9000H番地の内容)とが交換されて結果的にHLは元に戻り，スタック・トップには9000H, 9001H番地の内容が入ってい

ることになります(図2-24)。POPするときは

```
EX (SP), HL
LD (9000H), HL
POP HL
```

とすればOKです。

EX (SP), HLを使えば，先ほどの1バイトPUSH, POPも実現できます。

図2-24

④F, SP, PC各レジスタの値を得る

●Fレジスタ

フラグの値は16進数で表示しても，直感的にどのフラグが立っているかを判断することはできません。そこで各ビットをすべて表示することにします。

まずFレジスタだけを取り出します。それにはいったんAFとしてPUSHしておいてからBCにPOPします。こうするとPUSHしたときのFレジスタはCレジスタに入ります。次にCレジスタを1ビット

ずつ調べて1なら“1”を，0なら“0”を表示します。

リスト2-19を見てください。140行と160行でF→Cのロードをします。その間にCRLFで改行だけすませておきます。170行でそのフラグをAレジスタに戻し，180行でカウンタをセットしてから270行までのループに入ります。ループの名前はROTATEです。

190行からは**リスト1-13**の焼き直しです。ただ220行でLD A,“1”を使わずに単なるインクリメントにしたことと，CHPUTを呼び出す前のAレジスタの退避にスタックを使ったこと，およびDJNZでループしたことの3点を改良しました。

モニタのGコマンドで実行する前にXコマンドでFレジスタを好きな値に設定してみてください。その値のビットパターンが表示されます。

リスト2-19

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                       ORG      0D400H
110:                  ;
120:    00A2 =        CHPUT        EQU      00A2H
130:                  ;
140:    D400 F5                    PUSH     AF
150:    D401 CD17D4                CALL     CRLF
160:    D404 C1                    POP      BC
170:    D405 79                    LD       A,C
180:    D406 0608                  LD       B,8
190:    D408 0E30     ROTATE:      LD       C,'0'
200:    D40A 07                    RLCA
210:    D40B 3001                  JR       NC,ZERO
220:    D40D 0C                    INC      C
230:    D40E F5       ZERO:        PUSH     AF
240:    D40F 79                    LD       A,C
250:    D410 CDA200                CALL     CHPUT
260:    D413 F1                    POP      AF
270:    D414 10F2                  DJNZ     ROTATE
280:    D416 C9                    RET
290:                  ;
300:    D417 3E0D     CRLF:        LD       A,0DH
310:    D419 CDA200                CALL     CHPUT
320:    D41C 3E0A                  LD       A,0AH
330:    D41E CDA200                CALL     CHPUT
340:    D421 C9                    RET
```

●SPの値を得る

スタックのレベルを知りたくなったときにはSPの値が見られると便利です。SPの値を何かのペア・レジスタに直接ロードする命令はないので，工夫が必要です。

Z80の命令表を見て，SPの値自体を取り扱う命令がないか探してみてください。INC SPやDEC SPは使えませんね。(SP)となっているものはスタック・トップに入っている値のことなのでダメです。

このときはADD HL，SPという命令を使います。HLレジスタを0にクリアしておいてからこの命令を実行すると，HLにはそのときのSPの値が残ります。

●PCの値を得る

PCの値を他のレジスタにもってくるのは少々困難です。というのはPCを直接扱う命令がないからです。CALL命令だけが次に実行する命令のある番地を自動的にスタックに積む動作をするので，これを使うしかありません。

リスト2-20がその例です。120行でGETをコールしたとき，スタック・トップには120行のDEC命令の番地が入ります。180行でそれをHLに取り出し，180～190行でBCにコピーしてからスタックに戻します(210行)。リターンして120～140行でそのBCを3つ減らしています。こうするとBCに入っているPCの値がちょうど120行のCALL命令のアドレスになります。

実行するときはモニタでGD400，D406↵としてください。160行の

リスト2-20

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:   D400                     ORG      0D400H
110:                ;
120:   D400 CD07D4              CALL     GET
130:   D403 0B                  DEC      BC
140:   D404 0B                  DEC      BC
150:   D405 0B                  DEC      BC
160:   D406 C9                  RET
170:                ;
180:   D407 E3      GET:        EX       (SP),HL
190:   D408 44                  LD       B,H
200:   D409 4D                  LD       C,L
210:   D40A E3                  EX       (SP),HL
220:   D40B C9                  RET
```

RET命令の直前までくるとブレークして全レジスタの内容が表示されます。BCがD400Hになってますか？　これでPCの値が得られました。

アセンブラを使ってコーディングするなら，もっと簡単にPCの値を得られます。アセンブラには“$”で表わされる「ロケーション・カウンタ」というものがあって，これが使われた行（ソース上の）の命令の先頭番地がいつも入っています。すなわち

```
D400    LD  HL, $
```

という使われかたをすれば＄にはD400Hが（もちろんHLにも）入るのです。

ロケーション・カウンタを利用すれば，その行のPCは簡単に知ることができます。LD HL, ＄とするだけでその行がある番地をHLに得られるからです。

3. CALL(RST), RET命令

①JP命令とのちがい

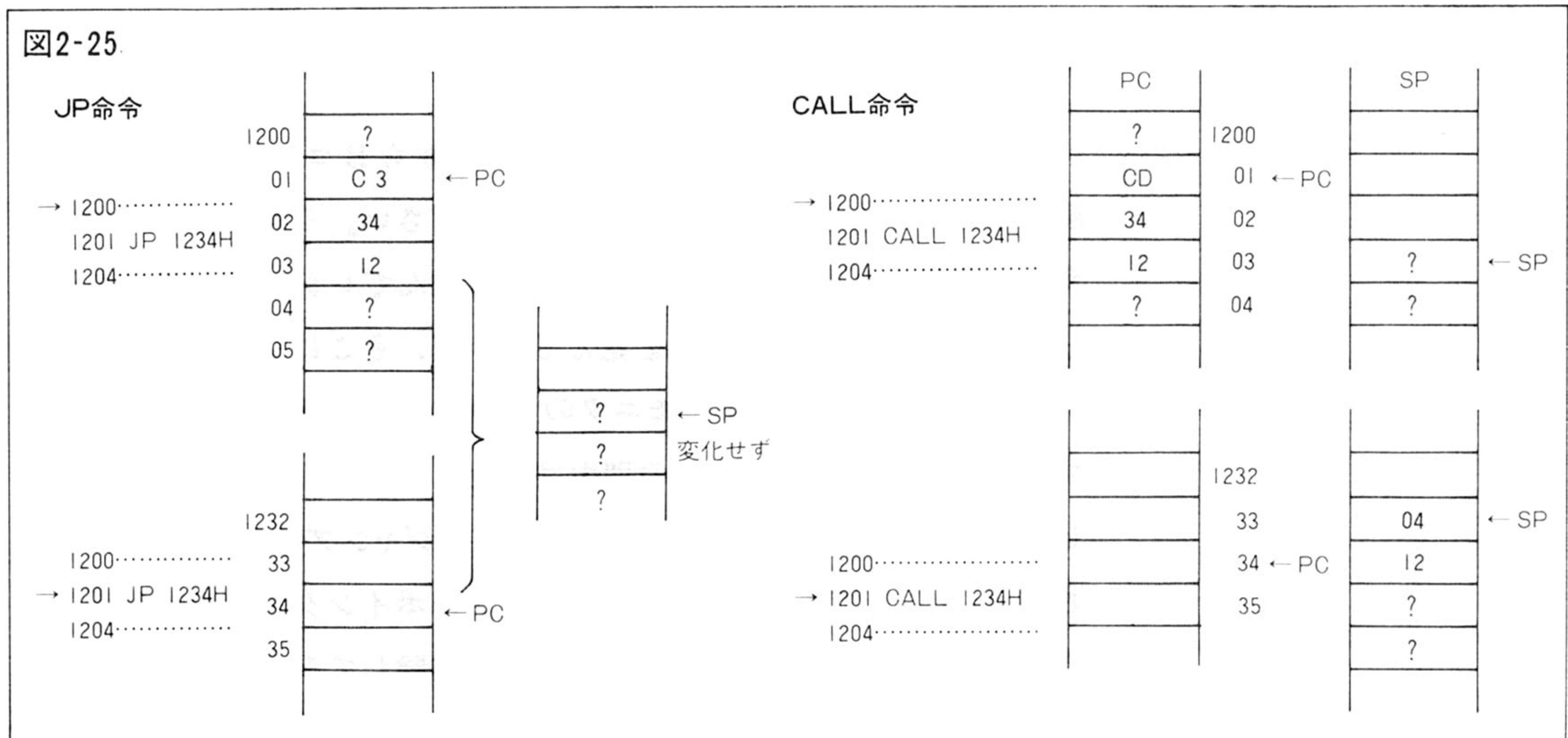
プログラムが現在実行している番地から別の番地へと実行を移させるのがジャンプ命令とコール命令です。現在実行中の番地から別の番地へ飛ぶときに，実行中の番地を保存しないのがジャンプ，保存するのがコールです。ジャンプ命令とコール命令が実行されるときのPC（プログラム・カウンタ）のようすを図2-25に示します。この図からわかることは①ジャンプ，コール命令とも，PCは次の命令が入っている番地を指している，②どちらの命令も実行によってPCにジャンプ先（またはコール先）が入る，③ジャンプ命令ではSPは変化しない，④コール命令では，コール命令の次の命令が入っている番地をSPの指す番地に格納してからコール先にジャンプする，ということです。

図2-25

JP命令

```
→ 1200……………       1200 | ?  |
  1201 JP 1234H       01  | C3 | ←PC
  1204……………         02  | 34 |
                      03  | 12 |
                      04  | ?  |
                      05  | ?  |
                                      | ? | ←SP
                                      | ? |  変化せず
                                      | ? |
  1200……………       1232 |    |
→ 1201 JP 1234H       33  |    |
  1204……………         34  |    | ←PC
                      35  |    |
```

CALL命令

```
                        PC                SP
→ 1200………………       | ?  | 1200        |    |
  1201 CALL 1234H     | CD | 01 ←PC      |    |
  1204………………       | 34 | 02          |    |
                      | 12 | 03          | ?  | ←SP
                      | ?  | 04          | ?  |

                      |    | 1232        |    |
                      |    | 33          | 04 | ←SP
  1200………………       |    | 34 ←PC      | 12 |
→ 1201 CALL 1234H     |    | 35          | ?  |
  1204………………                          | ?  |
```

つまりコール命令がジャンプ命令とちがうところは，コールのあと戻ってくるべき番地をスタックに記憶させておくということです。ジャンプ命令はジャンプしたあと戻る予定にはなっていないのに，コール命令は戻る予定でジャンプするのです。

戻るための命令はリターン(RET)ですが，RET命令はそのときのスタックのトップにある2バイトを戻り番地としてPCに入れているだけなのです。

②RETでジャンプ

このことを使うとRET命令でジャンプができます。**リスト2-21**を見てください。これを実行するとモニタのコマンド待ちに戻ります。

130行のEX (SP)，HLは現在SPが指しているメモリ(すなわちスタックのトップ)とHLレジスタの内容とを交換する命令です。HLには120行でEC00Hが入っているため，このEX命令でスタック・トップにEC00Hが入ります。140行のRET命令はそのときのスタック・トップをそのままPCに入れてジャンプするので，このときはPCにEC00Hが入り(EC00H番地にジャンプし)ます。EX命令を実行したときにHLレジスタに入るデータは本来RET命令で戻るべき番地です。

リスト2-21

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                    ORG     0D400H
110:                   ;
120:    D400 2100EC             LD      HL,0EC00H
130:    D403 E3                 EX      (SP),HL
140:    D404 C9                 RET
```

ただこのプログラムを実行するとモニタからBコマンドでBASICに戻ることができなくなるので注意してください。1回だけこのプログラムを実行し(GD400，D404)，モニタに飛んでレジスタが表示されたときのHLレジスタの中身を覚えておいて，そこにジャンプしないといけないのです。知らずにモニタのBコマンドを実行すると暴走します(しないこともある)。その理由はみなさんが考えてみてください。ヒントは，「モニタのGコマンドの実態はジャンプでなく，コールである」ということです。もっとも，スタックのポインタや内容が狂うと大変なことになるということを身をもって体験してみるのもよいでしょう。このように，EX (SP)，HLという命令は，スタックを操作する

には便利である反面危険な「両刃の剣」なのです。

また，コールしたときにSPに入れる番地はコール命令の次にある命令の番地だということをうまく利用する「インライン・パラメータ」などの手法もあります。このテクニックはのちの章で詳しく解説する予定です。

なお，**リスト2-21**の130行のEX命令をPUSH HLに代えると，もっとエレガントになります。これによってスタック・トップにはHLの内容(EC00H)が入るので，RETですなおにEC00Hにジャンプしていきます。この方法なら何度実行しても暴走しません。なぜならスタックのレベル(深さ；どれだけスタックにデータを積んだかということ)および内容が狂うことがないからです。

③ INC SP, DEC SP

スタック・トップを指すポインタ自体を動かすのがこの命令です。スタック・ポインタを動かすことはとても危険なことで，あまりひんぱんにしてよいことではありません。

SPを加減する必要があるのは，おもにスタック上のデータに何らかの操作，それも技巧的な操作をするときです。そうしたテクニックは以降で取り上げるので，そのときにこの命令の使いかたを説明します。

④サブルーチン，メイン・ルーチン間のデータ授受

メイン・ルーチンからサブルーチンを呼び出すとき，サブの動作内容によってはメインから何らかのデータを引き渡してやる必要があります。これがパラメータとか引数とか言われるものです。

たとえばMSXのBIOS(一覧表の一部を**表1-1**に掲載)には「エントリーパラメータ」と「リターン・パラメータ」の2種が明記されています。エントリーパラメータは，そのBIOSを呼び出すときに持っていくデータ，あるいはその入れもののことです。これまでよく使った1文字出力(00A2H)はAレジスタに表示したい文字のASCIIコードを入れてパラメータとしていました。

一方のリターン・パラメータは，BIOSから戻ったときに結果をどういう形で何に入れてくるかということを示します。キーボードの1文字入力(009FH)は，戻ってきたとき入力された文字のASCIIコードをAレジスタに入れています。

サブルーチンへのパラメータの渡し方法が上記のようなレジスタ経由なら，引き渡し方法にそれほど工夫はいりません。ところが，どうしても空いているレジスタがないとするとどうでしょう。

このようなときは，ふつうメモリ上にワーク・エリアをとって使います。メイン側で引き渡したいパラメータを所定のワーク・エリアにストアしておいてからサブルーチンを呼び出せばよいのです。パラメータが文字列であればそれが格納されているメモリ番地の先頭を指定するなどの場合もあります。

ただ，ワーク・エリアをとる方法はプログラム全体のサイズをエリア分だけ多くするという欠点があります。1度しか使わないパラメータのために数バイトのメモリを食われるというのは，場合によっては欠点ともなりうるのです。

ワーク・エリアをとる方法のほかに，スタックを使う方法があります。そこで

```
      LD HL, DATA
      PUSH HL
      CALL SUB
        :    :
SUB : POP   HL
        :    :
        :    :
```

などということを考えがちですが……。

実は上のようにするとおそらく暴走します。というのは，PUSHされたデータをサブ内でPOPしたつもりでも，HLレジスタに入るのはDATAではなくCALL命令の戻り先番地だからです。ここで戻り先番地を取り出してしまえば，スタック上に残っているのはもはやPUSHされたDATAだけ。これをサブの最後のRETで戻り先とまちがえてジャンプしたのではひとたまりもありません。

そこで，サブでDATAをまちがえなく取り出すために一計を案じねばなりません。もちろん戻り番地はRETのときにスタック・トップになければなりません。**リスト2-22**を見てください。メイン側の150行で入力した1文字をサブ側の230行でプリントします。200行のPOPは戻り先番地，210行のPOPはデータをそれぞれ取り出し，220行で戻り先だけをスタックに戻しておきます。

この方法で十分なのですが，**リスト2-22**のDISPサブルーチンを以下のようにしてもOKです。

```
DISP : POP HL
       EX (SP), HL
       LD A, H
       CALL PUTCH
       RET
```

これは，いったんPOPしたスタック・トップ(戻り先番地)をスタックの次のレベルに格納されているデータと交換する方法です。EX命令まで実行した段階でHLにはデータ(とフラグ)，スタック・トップには戻り先がそれぞれ収められています。LD A，Hでデータだけ(Hレジスタだけ)をAレジスタに移しています。

後者の方法のほうがメモリを操作することが少ないので，少しだけ速いです。

パラメータが2バイトにわたるときも当然この方法が使えます。

また，サブルーチン内で得たデータをメイン側に持ち帰る形のパラメータ授受もスタック経由で可能です。

リスト2-22

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                     ORG      0D400H
110:                  ;
120:    009F =        CHGET      EQU      009FH
130:    00A2 =        CHPUT      EQU      00A2H
140:                  ;
150:    D400 CD9F00              CALL     CHGET
160:    D403 F5                  PUSH     AF
170:    D404 CD08D4              CALL     DISP
180:    D407 C9                  RET
190:                  ;
200:    D408 C1       DISP:      POP      BC
210:    D409 F1                  POP      AF
220:    D40A C5                  PUSH     BC
230:    D40B CDA200              CALL     CHPUT
240:    D40E C9                  RET
```

⑤相対CALL

ジャンプ命令には，オペランドにジャンプ先のアドレスを直接書く絶対ジャンプと，そのジャンプ命令から前後に何バイト飛び越すかを示す数値(ディスプレイスメント，または変位という)をオペランドにする相対ジャンプとがあります。

たとえばD300H番地から実行されるプログラムがあったとします。このプログラム中で使われた命令がロードの大半や演算のような“アドレスと関係のない”命令ばかりだったら，このプログラム全体をD400H番地からにそっくり転送しても支障なく走ります。ところが絶対ジャンプやコール，LD HL，0D378Hなどのような“アドレスを参照する”命令が含まれていたらそうはいきません。同プログラム中で“CALL 0D378H”などとしているのを知らずに全体をD400H番地からに転送したら，D378H番地からの「もう何も入ってない」サブルーチンをコールしたとたんに暴走するのはうけあいです。

このようなことを防ぐため，Z80にはリロケータブル(どのメモリ・エリアに置いてもよい状態)にする命令があるのです。相対ジャンプはその1つです。ジャンプできる範囲が1バイトで表わせる数(0〜FFHまで)であることが難ですが，ちょっとしたプログラムなら間に合います。

しかしコールはいけません。BIOSコールのように，リロケートしたいプログラム中にない番地を呼び出すのなら絶対アドレスでもかまいませんが，同一プログラム中に含まれるサブルーチンをコールするとリロケータブルの夢は破れます。

そこで相対コール命令を自前で作ることにします。コール先のアドレスは相対ジャンプのような限られた範囲でなく，0〜FFFFHまで，Z80のメモリ空間すべてとしましょう。

リスト2-23がその一例です。140行〜160行がメインにあたり，180行〜200行までがSBRという名のサブルーチンです。メインからSBRをコールするのに220行〜のリロケータブル化ブロックを使います。RELと名づけられたリロケータブル化ブロックを使うときは，140行〜150行のようにまずRELをコールし，コール命令実行後に戻ってくるべき番地(RTN)とサブルーチンの番地＝コールしたい番地(SBR)との差をとってコール命令の3バイトの直後にDEFWで書き込んでおくのです。

メインではBIOSの1文字を使って‘H’という文字をプリントするだけです。

それではRELの動作を説明しましょう。P.82の**図2-26**を見てくださ

い。これはCALL RELを実行した直後からの各レジスタのようすを示しています。①は140行のCALLを実行してRELにきた直後の状態です。HLとDEには何が入っているかわかりません。スタックにはCALLしたときの戻り番地，すなわち140行のCALL命令の次の命令が入っている番地(D403H)がしまってあります。RELの220行でPOP HLを実行すると②のようにHLにその戻り番地が取り出せます。スタックは空になります。SPのところ（斜線になっている）には最後にBASICに戻るための番地が入っています。

260行までを実行すると③のようになります。HLが指す番地は150行のDEFWなので，SBRとRTNとの差がDEに入ります。先にEからロードしているのはDEFW擬似命令が上位と下位を逆(0001が0100となる)にしてメモリに格納しているからです。260行でHLを＋1するとHLの内容は160行のD405H番地を指しています。

270行でHLをスタックに入れます(④)。このときのHLの内容(D405H＝RTN)はSBRをコールしたあとで戻るべき番地です。①と④

リスト2-23

```
 MSX Self Assembler  Rev 1.0     PAGE    1


100:    D400                     ORG     0D400H
110:                   ;
120:    00A2 =         CHPUT     EQU     00A2H
130:                   ;
140:    D400 CD0CD4              CALL    REL
150:    D403 0100                DEFW    SBR-RTN
160:    D405 C9        RTN:      RET
170:                   ;
180:    D406 3E48      SBR:      LD      A,'H'
190:    D408 CDA200              CALL    CHPUT
200:    D40B C9                  RET
210:                   ;
220:    D40C E1        REL:      POP     HL
230:    D40D 5E                  LD      E,(HL)
240:    D40E 23                  INC     HL
250:    D40F 56                  LD      D,(HL)
260:    D410 23                  INC     HL
270:    D411 E5                  PUSH    HL
280:    D412 19                  ADD     HL,DE
290:    D413 E5                  PUSH    HL
300:    D414 C9                  RET
```

とでスタックの内容を比べると戻り番地が＋2されている（DEFWの2バイトをスキップしている）のがわかりますね。'2つのINC命令のおかげです。

280行でHLとDEを加えているのがこのプログラムのミソです。このときのHLは相対コールを実行したあと戻る番地(RTN)，DEはそこから目的のサブルーチン(SBR)までの距離(DEFWの内容)ですね。

図2-26

①220行にきた直後

スタック

HL: ? ?

DE: ? ?

スタック: 03 ←SP, D4

②220行を実行

スタック

HL: D4 03

DE: ? ?

スタック: ←SP

③260行までを実行

スタック

HL: D4 05

DE: 00 01

スタック: ←SP

④270行を実行

スタック

HL: D4 05

DE: 00 01

スタック: 05 ←SP, D4

⑤290行までを実行

スタック

HL: D4 06

DE: 00 01

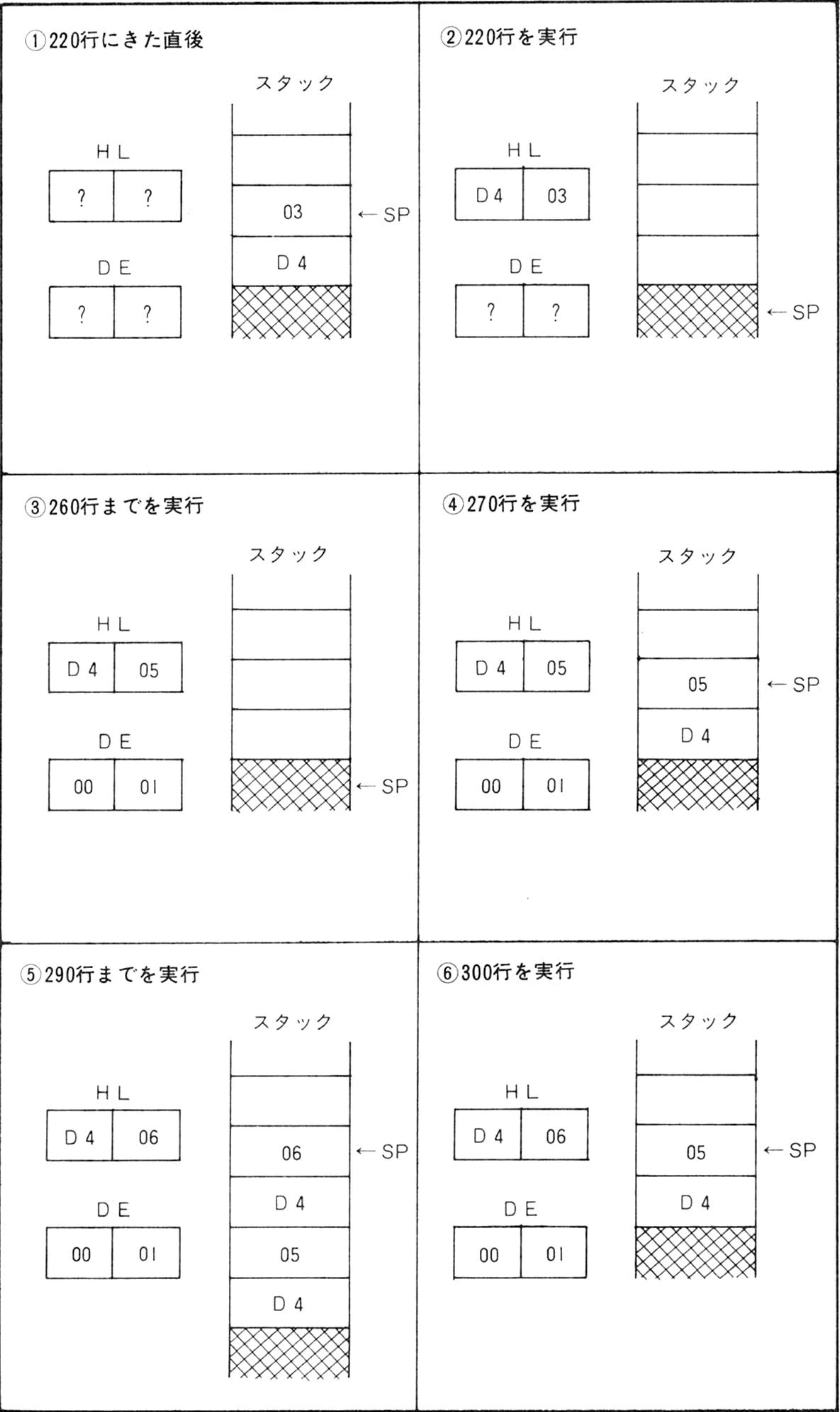
スタック: 06 ←SP, D4, 05, D4

⑥300行を実行

スタック

HL: D4 06

DE: 00 01

スタック: 05 ←SP, D4

この関係はZ80の相対ジャンプ(JR)命令における命令の位置とディスプレイスメントとの関係に相似しています。280行でこれらを加えれば，結果として残るHLの内容はコール先(SBR)の番地となるのです。これを290行でスタックにしまいます。ここまでが⑤です。

300行でRETを実行するとどうなるでしょう。⑤にあるようにスタックの先頭は290行でPUSHしたHLの値，すなわちコール先SBRの番地(D406H)です。RETはこれをプログラム・カウンタに入れてジャンプしますから，RETによって制御はSBRに移りますね(⑥)。SBRのサブルーチンは以上のような経緯で実行されるに至ったわけです。

サブルーチンの実行を終えて200行のRETにつき当たると再びスタックから戻り番地を取り出します。このときのスタックは⑥の状態になっているため，D405H番地(RTN)に戻っていくのです。

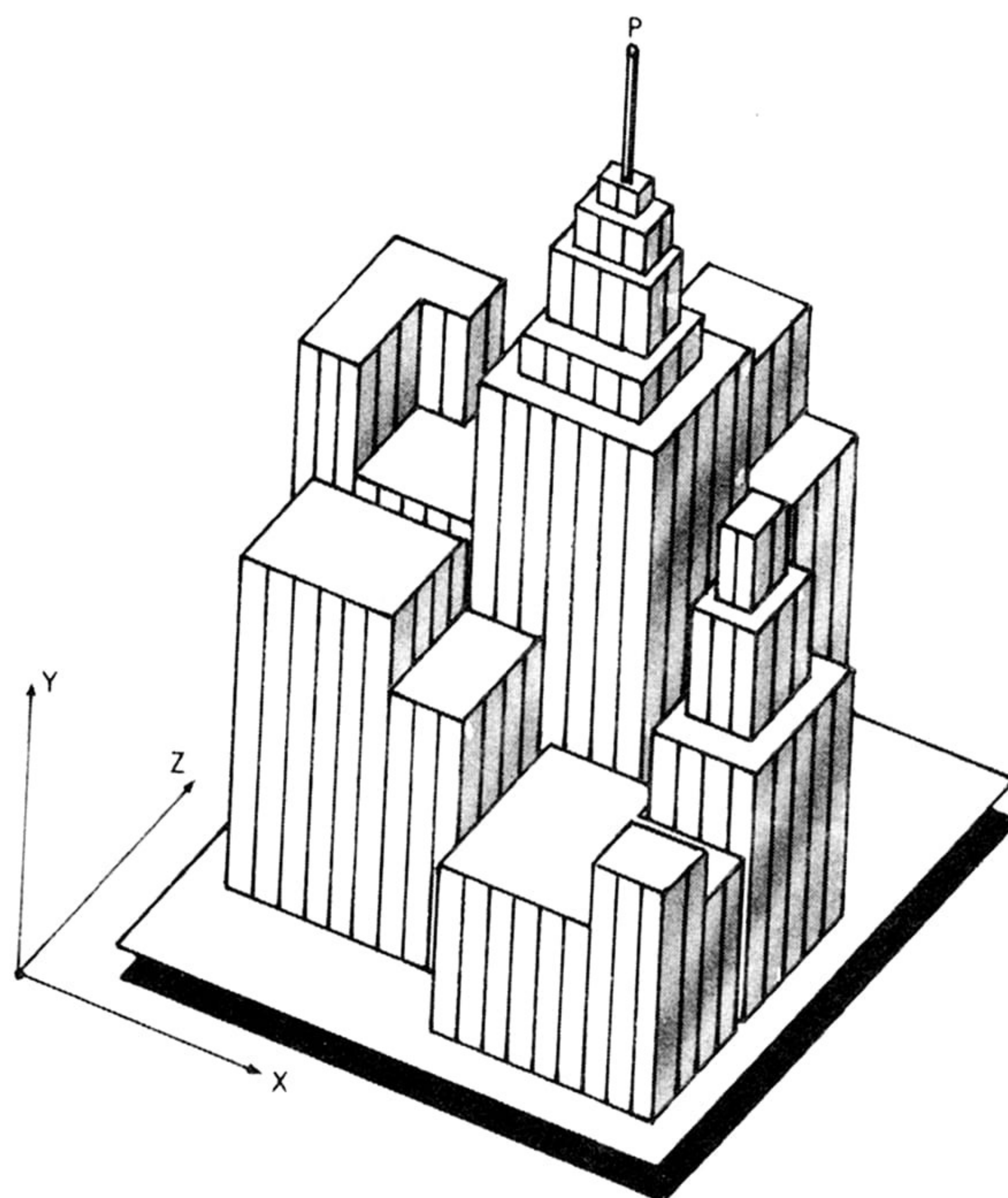
長ったらしい説明になってしまいました。おわかりいただけたでしょうか。この相対コールを使うときは140行～160行のようにしてください。SBRは目的のサブルーチンの番地，RTNは必ずDEFWの直後，DEFWは必ずCALLの直後でなくてはなりません。

この方法で目的のプログラムのコール命令をすべて相対化したら，そのプログラムはリロケータブルになるでしょうか。もちろんJP命令など使わないとしてです。答は条件つきでYESです。その条件とは，「REL以下の9バイトはリロケートできない」ということです。**リスト2-23**でいえば，D400H～D40BH番地まではどこに転送してもよいのですが，D40CH～D414H番地までは動かせません。

それはRELルーチン自体に原因があるのではありません。RELルーチンの9バイトには絶対アドレスを操作する命令を含んでいません。原因は140行にあります。CALL RELがクセモノで，RELという番地はどうしても絶対アドレスなのです。もしD400H～D414H番地までをそっくり別のアドレスに移したとすると(たとえばD300H～)，RELをコールするつもりで140行のオブジェクトのようにCALL D40CHとやってしまうのです。

というわけで，REL以下の9バイトはBIOSのようにどこか一定の空きアドレスに固定しておかねばならないのです。もしREL以外の部分が何Kバイトもある大きなプログラムであれば，REL以下の9バイトは微々たるものといえるでしょう。**リスト2-23**のようだと少々もったいないですね。

コール命令を条件つきで相対化してみましたが，もっとスマートな

方法があるかもしれません。また，大きなプログラムではJP命令も使うことが多いので，JP命令も64Kバイトのメモリ空間で相対化しないと実用にならないでしょう。

3章 基本テクニックをまとめてみよう

実践テクニックを覚えたら腕だめしだ。扱うのはマシン語の顔，ダンプ・コマンド。自分で設計すれば機能拡張も難しくない。BASICで書けば簡単なプログラムも，マシン語だとこんなに詳しく書かなければならないということを，じっくりごらんください。

これまで，いくつかの基本テクニックをみてきました。この章ではまとめとして，これまでに例としてあげたプログラムよりももっとまとまった動作をさせてみましょう。画面に動作内容がすぐ反映してわかりやすいということで，「メモリ・ダンプ」ルーチンをつくります。

実際にある程度大きなプログラムを開発しようとするとき，一定の手順を踏んでいったほうが無駄なく進行します。ここではメモリダンプ・ルーチンをつくることを目標に，本書に兄貴分である「マシン語入門・基礎編」の第5章「マシン語プログラムの作成方法」を参考にして作成をすすめます。

1 基本テクニックをまとめてみよう
問題を解析し，仕様を決定する

メモリ・ダンプの動作については，すでにご存知のことと思います。モニタのDコマンドなどを使えば，任意の番地のメモリ内容を見ることができますね。もしまだメモリ・ダンプというものを見ていないなら，すぐしてみてください。手元にダンプできるモニタがなければ，各マイコン関係誌に掲載されたマシン語リスト（ダンプ・リスト）を思い浮かべていただければ十分です。

〔問題〕
- プログラム名は「メモリダンプ・プログラム」。
- ">"印を1つプリントし，キーボードからダンプする範囲を指定する。指定は16進数で表現された2つの番地とし，それぞれダンプ開始アドレス，ダンプ終了アドレスとする。2つの番地はカンマで区切る。
- 入力に16進数文字以外の文字があったり，開始アドレス＞終了アドレスだったときはエラーである旨表示して再入力を促す。
- ダンプは1行8バイトとし，チェックサムやASCII文字は出力しない。

以上のように問題を規定し，細部の仕様について検討します。

〔プログラムの仕様〕

1．キーボードからの入力，画面への表示など，BIOSを使って実現できるものはできるだけBIOSを使う。

2．入力はBIOSの1行入力を使う。入力できる文字は'0'～'9'までの数字と'A'～'F'までの英大文字とする。2つのアドレス間はカンマ(，)で区切り，アドレスの一方，または両方とも省略できる。

3．1行入力を終えた時点で入力文字列を解析し，開始アドレスと終了アドレスをいったんメモリ内のワーク・エリアに格納する。開始アドレスが省略されていたら直前に表示したときの終了アドレス＋1番地から，また終了アドレスが省略されていたら開始アドレスから128バイト(16行)分とする。両方とも省略されていたときもこれに準ずる。どちらのワーク・エリアも初期値はゼロとする。

4．ダンプ中にCTRL＋STOPキーが押されたときはダンプを中断し，モニタのコマンド待ちに戻る。

5．入力待ち，入力中ともCTRL＋STOPキーで中断し，BASICのコマンド待ちに戻ることができる。

6．プログラム中で画面のクリアはしない。

7．入力文字列に16進文字以外の文字が含まれていた場合および開始番地>終了番地の関係になっていた場合は改行して'?'を表示し，再び入力に戻る。

8．ダンプを正常終了したら再び入力に戻る。

9．このプログラムはモニタのGコマンド，またはBASICのUSR関数で実行する。プログラムの実行アドレスはD300H番地とする。

以上のような仕様に決めます。実用的なルーチンなら，ダンプ中に何かのキーで表示を一時停止できる，プリンタへの出力ができる，チェックサムやASCIIコードの出力を同時にする，などの機能もつけるべきでしょうが，今回のメモリ・ダンプは学習用ということでガマンしています。みなさんが実力アップのために改造してみてください。

基本テクニックをまとめてみよう

2 プログラムの概要設計

決定した仕様を見て，ゼネラル・フローチャートを書きます。ゼネラルですから処理内容の表現は大まかでよく，とにかく後のためにわかりやすく書きましょう。図3-1のようになりました。結局コマンド入力部とダンプ表示部の2つに分けましたが，最初からこれほど細かく書くのは無理かもしれません。初めは右の図くらいの粗さで，これをどんどん細かくしてゆくのがよいでしょう。

START
コマンド入力
ダンプ表示
END

図 3-1 のゼネラル・フローチャートについて，少し説明しておきます。(1)コマンド入力部(KEYIN)は，BIOS の1行入力で1行分のコマンド列を入力します。このBIOS(PINLIN=00AEH)は，リターン・キーか CTRL-STOPキーが入力されるまで，画面に表示しながら1行分をまとめて1行入力バッファと呼ばれるワーク・エリア(F55EH～)に入れるものです。PINLINから戻ってきたとき，HLにはバッファの先頭－1(F55DH)が入っています。もし1行入力がCTRL-STOPで終了したときは，戻ってきたときCフラグが立っています。バッファ内はASCIIコードで文字列が格納されており，入力の終了(リターン・キーとかCTRL STOPキーが押された）は00で表わされています。

したがって，1行入力のあとでCフラグを見ればSTOPキーが押されていたかの判断ができるわけです。もし押されていたならメモリダンプ・ルーチンを中断してBASICに戻ります。

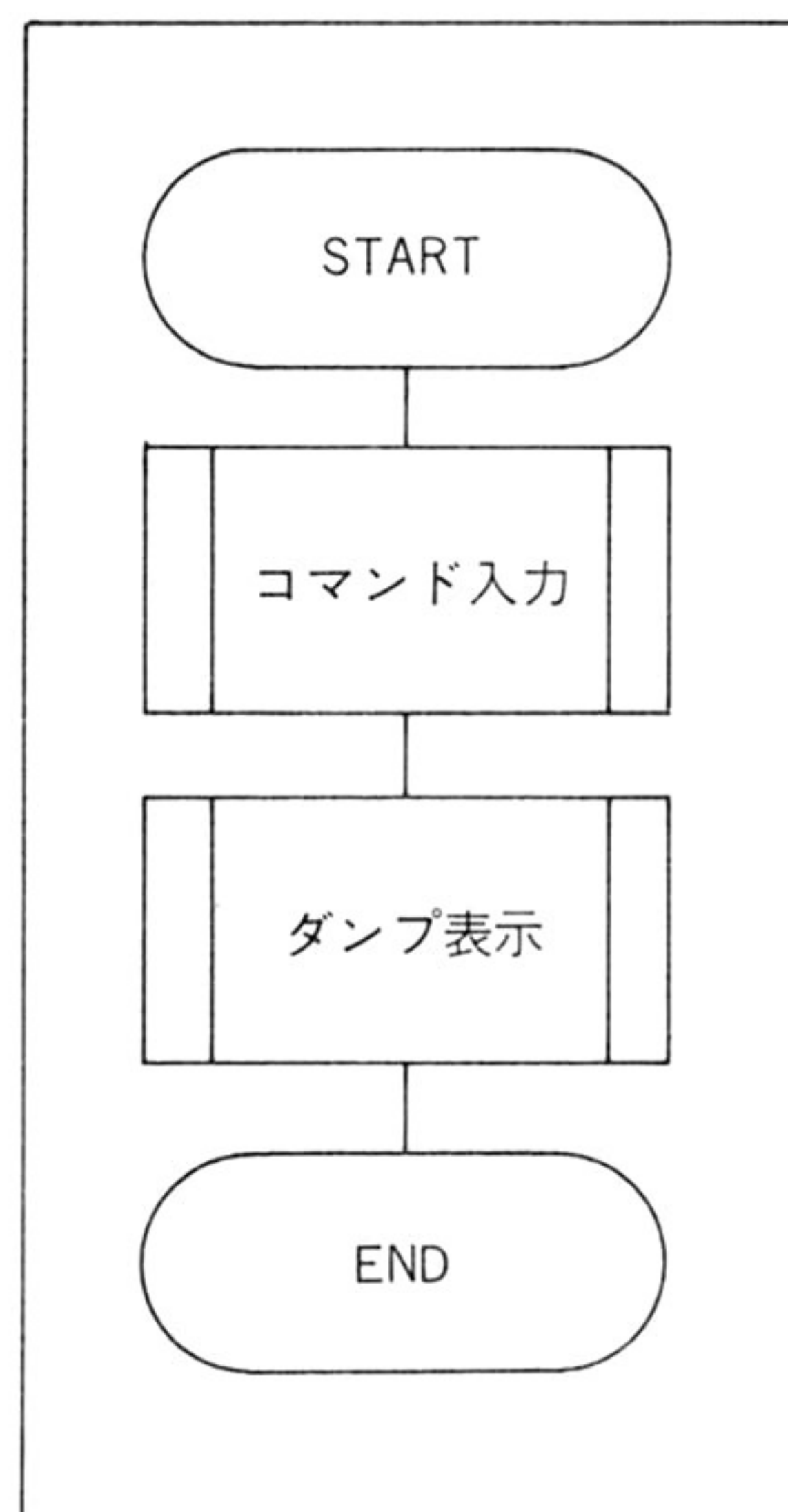
コマンド列の解析は，1行入力されたバッファの内容を1文字ずつ調べています。コマンドとして予想しているのは

① ，RETURN

② ，8000RETURN

③ 7000，8000RETURN

④ 7000 RETURN

⑤ 7000，RETURN

の5通りに加えて，これらの16進数列が1～多数桁におよぶ場合です。5桁以上なら最後の4桁を有効にします（例：1234567890　と入力されたら7890と解釈する)。これら以外の入力はエラーとなります。

開始アドレスが省略されたら，直前にこのルーチンでダンプした終了アドレス＋1を新しい開始アドレスとします（つまり直前のダンプのつづきになる)。終了アドレスが省略されたら，開始アドレスから，128バイト分とします。

フローチャート中で「ASC→HEX」と書かれているところは，たとえば文字列の“ABC”を16進数のABCHに変換することを表わしています。ASCⅡ文字列から16進数への変換ということです。

ダンプ表示部(DUMP)は，もっと構造がすっきりしています。KEYIN部で16進数文字のチェックをすませて16進数に変換した数がFROMとTOで表わされたワーク・エリアに格納されてきているため，それを

図3-1 ゼネラル・フローチャート

(1)コマンド入力“KEYIN”，
FROMに開始アドレス，
TOに終了アドレス

KEYIN
プロンプト表示
1行入力　BIOS使用
STOP有？　Y → BASICにRET
N
開始アドレス省略？　Y → FROM＝以前のTo＋1 → ①
②→ N
16進数？　N → ERR
Y
ASC→HEX　開始アドレス
次はカンマ？　N → ②N リターンキー？　Y → FROM＝開始アドレス TO＝FROM＋127 → DUMPへ
Y
FROM＝開始アドレス
←①
次は16進数？　N → リターンキー？　Y → TO＝FROM＋127 → DUMPへ
リターンキー？ N → ERR
Y
ASC→HEX
次はリターンキー？　N → ①
Y
TO＝終了アドレス
DUMPへ

ERR
改行して'?'を表示
KEYIN

(2)ダンプ表示“DUMP”，
FROM～TOまでを
画面表示

DUMP
FROM≦TO？　N → ERR
Y
アドレス表示
データ1バイト表示
スペース1つ表示
FROM＝FROM＋1
FROM＞TO？　N → KEYINへ
Y
1行8バイト表示した？　N → (データ1バイト表示へ)
Y
改行
STOPキー押された？　N → (アドレス表示へ)
Y
モニタへ

元にして文字を表示するループだけですんでいるからです。

まず指定されたアドレスFROMとTOの関係を調べます。終了アドレスより開始アドレスのほうが大きくては異常ですから，エラー処理に行きます。そのチェックがＯＫならアドレス部分を表示し，１行８バイトずつ，１バイトごとに１つスペースを空けながら表示します。１バイト表示したらFROMをインクリメントし，FROMがTOを越えたかどうか判定します。越えていたらダンプ終了，KEYINから再スタートです。

１行分の表示が終わったら改行し，またアドレスの表示から次の１行分を表示します。CTRL-STOPが押されたかどうかはBIOS(BREAKX＝00B7H)を使って，１行分を表示した時点で調べます。BREAKXはCTRL-STOPが押されていたらCフラグを立てて戻ってきます。もしCTRL-STOPが押されていたらモニタ(EC00H～)にジャンプします。

「エラー処理」のルーチンは，単に'?'をプリントし，KEYINからやりなおすだけです。

3 基本テクニックをまとめてみよう プログラムの詳細設計

つぎに，できたゼネラル・フローチャートをコーディング（実際にZ80のモモニックでプログラムを書くこと）ができるようになるまで細かく記述しなおします。これをディテール・フローチャートといいます。

今回のメモリ・ダンプはかなりたくさんのディテール・フローチャートになってしまいました P92，93 の図3-2，4)。

P92の図3-3は１行入力バッファ，FROM，TOなどのワーク・エリアの内容，表3-1はサブルーチン一覧表です。

ディテール・フローチャートのうち，説明を要するものが少々あります。まずKEYINで1行入力をしてから，バッファ上を指すポインタのDEレジスタを＋２しているのは，もともとこのBIOS(PINLIN)から戻ってきたときにはHLがバッファの先頭－１になっているのと，プロンプト（>）を１文字表示しているためです（図3-3参照)。開始アドレス，終了アドレスとも，それを16進数に変換するときHLに入れられるのですが，プログラム開始時点でHLをクリアしておかない

と，たとえば'10，20'などと4桁未満で入力したときに正しく10や20に認識されません。

16進数で使える文字以外の文字が現われたときは，開始・終了アドレスの区切りとなるカンマ（，）か，入力の終了を示すリターン・キー（1行入力バッファ上では00で表わされている）かを判定しています。これら以外はエラーとなります。

ASC↔HEXで変換するとき，Aレジスタに7を加えたり加えなかったりしているのは，文字'9'と'A'のASCIIコードの間が連続していないからです。このことはASCIIコード表をにらんでいるとわかるでしょう。

自分（作成者）以外の他人が見て使うためでなければ，ディテール・フローの書きかたは作成者本人のわかりやすい方法でかまわないでしょう。

表3-1　サブルーチン一覧表（＊印はBIOS）

ラベル名	エントリーパラメータ	リターンパラメータ	変化するレジスタ	内　容
＊ BREAKX	なし	押していたらCフラグON	なし	CTRL-STOPキーが押されているか調べる
＊ CHPUT	Aレジスタに文字コード	なし	なし	画面に1文字表示
＊ PINLIN	なし	HLレジスタにバッファの先頭番地－1が入る。STOPを入力したときはCフラグがON	すべて	RETURNやSTOPを入力するまでに入れた値をラインバッファに入れる
ADRP	DEは表示するアドレスを表わす16進数4桁	なし	A, BC, DE	DEの内容をASCII4文字として表示，つづけてスペースを1つ表示
CHKHEX	Aレジスタに調べる文字のコード	16進文字ならCフラグOFF 16進文字ならCフラグON	AF	Aの内容が16進数として許される文字かどうかを調べる
CRLF	なし	なし	A	画面にCR，LFコードを出力する（改行する）
DATAP	DEは表示するデータが格納されたアドレス	なし	A, DE	（DE）で表わされる番地のメモリ内容をASCII2文字として表示，つづけてスペースを1つ表示
TOASC	AにASCIIコードに変換したい1バイトの数値	BCに変換された2文字のASCIIコード	BC	Aの内容を16進数とみてASCII文字2文字に変換BCレジスタに入れる
TOHEX	Aに16進数に変換する文字のコード	HLの最下位4ビットに変換された16進数	HL	Aの内容を16進数文字のASCIIコードとみてHLの最下位からにつけ加える

図3-2

'DUMP

DUMP
DE←FROM
HL←TO
DCOMPR　DEとHLを比較
DE>HL?　Y → ERR
N
ADRP　アドレス部表示
B←8　1行8バイトのカウンタ
DATAP　データ部1バイト表示
DE=DE+1　次の1バイト
DCOMPR　FROM>TO?
DE>HL?　→ KEYIN
N
B←B−1
B=0?　N
DJNZ
Y
CRLF　改行
BREAKX　CTRL−STOP チェック
STOP?　Y → MON
N

KEYIN(2)

③ → FROM← TO+1
① → ・,・?　N → ↵?　Y → FROM←HL TO←FROM+127 → DUMP
↵? N → ②
Y
FROM←HL
A←(DE+1)　次の文字
←HL=0
CHKHEX
HEX?　N → ↵?　Y → TO←FROM +127 → DUMP
↵? N → ERR
Y
TOHEX
A←(DE+1)　次の文字
↵?　N
Y
TO←HL
DUMP

ERR
A←"?"
CHPUT
KEYIN

KEYIN(1)

KEYIN
CRLF
CRLF
A←">"
CHPUT　プロンプト出力
PINLIN　1行入力
STOP?　Y → BASICにRET
N
DE←HL　1行入力バッファー1をDEに
DE=DE+2　プロンプト分をスキップ
A←(DE)
・,・?　Y → ③
N
HL←00　クリア
② →
CHKHEX　16進文字か？
HEX?　N → ERR
Y
TOHEX　HL←HEX
DE=DE+1　次の文字
A←(DE)
①

図3-3 ワーク・エリアの内容

●1行入力バッファ（例：7000, 8000と入力）

2C	3E	37	30	30	30	2C	38	30	30	30	00	00	00	00	………………………
(,)	(>)	(7)	(0)	(0)	(0)	(,)	(8)	(0)	(0)	(0)	区切りのゼロ				

BUF−1
BUF(F55EH) プロンプトが入っている。
F567H
PINLINから戻ってきたとき，HLはここを指している。内容は必ずカンマ。

●FROM, TO

00	00	FROM
00	00	TO

ともに初期値はゼロ。

図3-4

●ADRP

- ADRP
- A←H
- TOASC（Aの内容をASCII2文字にしてBCに入れる）
- A←B
- CHPUT
- A←C
- CHPUT
- A←L
- TOASC
- A←B
- CHPUT
- A←C
- CHPUT
- A←" "（スペース）
- CHPUT
- RETURN

●TOASC

- TOASC
- スタック←A
- A←A AND F0H（上位4ビットを残してマスク）
- TOASC2
- B←A
- A←スタック
- スタック←A
- A←A AND 0FH（下位4ビットを残してマスク）
- TOASC3
- C←A
- A←スタック
- RETURN

●DATAP

- DATAP
- スタック←B
- A←(DE)
- TOASC
- A←B
- CHPUT
- A←C
- CHPUT
- A←" "（スペース）
- CHPUT
- B←スタック
- RETURN

●TOHEX

- TOHEX
- スタック←A
- '0'～'9'?（Y: A←A-7を飛ばす／N: A←A-7）
- A←A-7
- A←A AND 0FH（下位4ビット残してマスク）
- HL←HL×16
- A←A OR L
- L←A
- A←スタック
- RETURN

●TOASC2

- TOASC2
- Aレジスタ右シフト（×4回）
- TOASC3

●TOASC3

- TOASC3
- A≧0AH?（N: A←A+7を飛ばす／Y: A←A+7）
- A←A+7
- A←A+30H
- RETURN

●CHKHEX

- CHKHEX
- A≧'0'?（N: Cフラグ←1／Y: 次へ）
- A<'9'+1?（Y: Cフラグ←0／N: 次へ）
- A≧'A'?（N: Cフラグ←1／Y: 次へ）
- A<'F'+1?（Y: Cフラグ←0／N: Cフラグ←1）
- Cフラグ←1
- Cフラグ←0
- RETURN

●CRLF

- CRLF
- A←CRコード
- CHPUT
- A←LFコード
- CHPUT
- RETURN

基本テクニックをまとめてみよう

4 コーディング

③で完成したディテール・フローチャートをもとに，アセンブラを使ってZ80のニモニックを打ち込みます。ディテール・フローチャートの各項目には，そのまま1つのマシン語命令に置きかわるものと数個の命令に展開されるものとがあります。打ち込む前に紙に書く作業のことをコーディングといい，コーディングが完了した時点で初めてMSXに向かって打ち込むことになるでしょう。打ち込むときはわかりやすくコメントをつけておくと，あとで便利です。打ち込み終わったら必ずテープにセーブします。

メモリダンプ・ルーチンをコーディングしたのが，リスト3-1です。

リスト3-1

```
  MSX Self Assembler  Rev 1.0


1000:                       ;
1010:                       ;MSX machine-language ﾆｭｳﾓﾝ No.3 (ｼﾞｯｾﾝ ﾍﾝ)
1020:                       ;
1030:                       ;  sample program
1040:                       ;
1050:                       ;   'MEMORY-DUMP routine'
1060:                       ;
1070:                       ;      by KEN , '84/5/11(FRI)
1080:                       ;
1090:                       ;
1100:                       ;
1110:    D300                         ORG     0D300H
1120:                       ;
1130:                       ;
1140:                       ; ﾗﾍﾞﾙ ﾃｲｷﾞ
1150:                       ;
1160:                       ;
1170:    00A2 =             CHPUT     EQU     00A2H  ;1ﾓｼﾞ ｼｭﾂﾘｮｸ BIOS
1180:    00AE =             PINLIN    EQU     00AEH  ;1ｷﾞｮｳ ﾆｭｳﾘｮｸ BIOS
1190:    00B7 =             BREAKX    EQU     00B7H  ;CTRL-STOP ｷｰ ﾆｭｳﾘｮｸ ﾁｪｯｸ BIOS
1200:                       ;
1210:    000D =             CR        EQU     0DH    ;CR ｺｰﾄﾞ
1220:    000A =             LF        EQU     0AH    ;LF ｺｰﾄﾞ
1230:                       ;
1240:    EC00 =             MON       EQU     0EC00H ;MSX-MONITOR ｲﾘｸﾞﾁ ﾊﾞﾝﾁ
1250:                       ;
1260:                       ; ﾒｲﾝ START
1270:                       ;   ｺﾏﾝﾄﾞ 1ｷﾞｮｳ ﾆｭｳﾘｮｸ 'KEYIN'
1280:                       ;
1290:    D300               KEYIN:
1300:    D300 CDDED3                  CALL    CRLF   ;ｶｲｷﾞｮｳ
1310:    D303 CDDED3                  CALL    CRLF
1320:    D306 3E3E                    LD      A,'>'  ;PROMPT
1330:    D308 CDA200                  CALL    CHPUT  ;1ﾓｼﾞ ﾋｮｳｼﾞ
1340:    D30B CDAE00                  CALL    PINLIN ;1ｷﾞｮｳ ﾆｭｳﾘｮｸ
```

```
1350:   D30E D8                  RET     C       ;STOPｷｰ ｶﾞ 1ｷﾞｮｳ ﾉ ﾅｶﾆ ﾊｲｯﾃ ｲﾀ
1360:                                            ; ｺﾉﾄｷ ﾊ BASIC ﾆ ﾓﾄﾞﾙ
1370:   D30F EB                  EX      DE,HL   ;1ｷﾞｮｳ ﾆｭｳﾘｮｸ ﾊﾞｯﾌｧ - 1 ｦ
1380:                                            ; ﾎﾟｲﾝﾀ ﾄｼﾃ DE ﾆ ｲﾚﾙ
1390:   D310 13                  INC     DE
1400:   D311 13                  INC     DE      ;PROMPT ﾏﾃﾞ SKIP ｽﾙ
1410:   D312 1A                  LD      A,(DE)  ;ｻｲｼｮ ﾉ ﾓｼﾞｺｰﾄﾞ ｦ Aﾚｼﾞｽﾀ ﾍ
1420:   D313 FE2C                CP      ','     ;DUMP_START ADDR. ｼｮｳﾘｬｸ ?
1430:   D315 2823                JR      Z,SECOND ;YES
1440:   D317 210000              LD      HL,0    ;HL(DUMP_START ADDR. ｶﾞ
1450:                                            ; ﾊｲﾙﾄｺﾛ) ｦ ｸﾘｱ
1460:   D31A            FIRST:
1470:   D31A CDE9D3              CALL    CHKHEX  ;16ｼﾝｽｳ ﾓｼﾞ ?
1480:   D31D DAC5D3              JP      C,ERR   ;NO -> ｴﾗｰ
1490:   D320 CDCDD3              CALL    TOHEX   ;16ｼﾝｽｳ ﾆ ﾍﾝｶﾝ
1500:   D323 13                  INC     DE      ;ﾂｷﾞ ﾉ ﾓｼﾞ
1510:   D324 1A                  LD      A,(DE)
1520:   D325 FE2C                CP      ','     ;ｸｷﾞﾘ ﾓｼﾞ ｶ ?
1530:   D327 CA3ED3              JP      Z,SECON2 ;YES
1540:   D32A FE00                CP      0       ;ﾃﾞﾊ ｺﾏﾝﾄﾞ ｼｭｳﾘｮｳ ｶ·?
1550:   D32C 20EC                JR      NZ,FIRST ;ﾃﾞﾓﾅｹﾚﾊﾞ 16ｼﾝｽｳ ﾓｼﾞ ﾄｼﾃ ｼｮﾘ
1560:   D32E 221FD4              LD      (FROM),HL ;RETURN ｷｰ ﾅﾗ KEYIN ｼｭｳﾘｮｳ,
1570:                                            ; FROM <- HL
1580:   D331 017F00              LD      BC,127  ;RETURN ｷｰ ﾅﾉﾃﾞ END_ADDR. ｶﾞ
1590:                                            ; ｼｮｳﾘｬｸ ｻﾚﾀ ﾄ ﾐﾅｽ
1600:   D334 09                  ADD     HL,BC   ;TO=FROM + 127
1610:   D335 2221D4              LD      (TO),HL ;TO <- HL
1620:   D338 182D                JR      DUMP    ;DUMP ｶｲｼ
1630:                   ;
1640:   D33A            SECOND:
1650:   D33A 2A21D4              LD      HL,(TO) ;START ADDR. ｼｮｳﾘｬｸ ﾉ ﾄｷﾊ
1660:   D33D 23                  INC     HL      ;FROM=TO + 1 ﾄｽﾙ
1670:   D33E            SECON2:
1680:   D33E 221FD4              LD      (FROM),HL ;FROM <- HL
1690:   D341 13                  INC     DE      ;ﾂｷﾞ ﾉ ﾓｼﾞ
1700:   D342 1A                  LD      A,(DE)
1710:   D343 210000              LD      HL,0    ;HL(END-ADDR. ｶﾞ ﾊｲﾙﾄｺﾛ)ｦ ｸﾘｱ
1720:   D346            SECON3:
1730:   D346 CDE9D3              CALL    CHKHEX  ;16ｼﾝｽｳ ﾓｼﾞ ?
1740:   D349 3010                JR      NC,SECON4 ;YES
1750:   D34B FE00                CP      0       ;ﾃﾞﾊ ｺﾏﾝﾄﾞ ｼｭｳﾘｮｳ ｶ ?
1760:   D34D 2076                JR      NZ,ERR  ;ﾃﾞﾓﾅｹﾚﾊﾞ ERROR
1770:   D34F 2A1FD4              LD      HL,(FROM) ;RETURN ｷｰ ﾀﾞｶﾗ END_ADDR.
1780:                                            ; ｼｮｳﾘﾔｸ ﾄ ﾐﾙ
1790:   D352 017F00              LD      BC,127  ;TO=FROM + 127
1800:   D355 09                  ADD     HL,BC
1810:   D356 2221D4              LD      (TO),HL ;TO <- HL
1820:   D359 180C                JR      DUMP    ;KEYIN ｼｭｳﾘｮｳ,DUMP ﾍ
1830:
1840:   D35B            SECON4:
1850:   D35B CDCDD3              CALL    TOHEX   ;16ｼﾝｽｳ ﾆ ﾍﾝｶﾝ
1860:   D35E 13                  INC     DE      ;ﾂｷﾞ ﾉ ﾓｼﾞ
1870:   D35F 1A                  LD      A,(DE)
1880:   D360 FE00                CP      0       ;ｺﾏﾝﾄﾞ ｼｭｳﾘｮｳ ｶ ?
1890:   D362 20E2                JR      NZ,SECON3 ;NO
1900:   D364 2221D4              LD      (TO),HL ;ｺﾏﾝﾄﾞ ｼｭｳﾘｮｳ,TO <- HL
1910:                   ;
1920:                   ;
1930:                   ; DUMP ﾙｰﾁﾝ
1940:                   ;    (FROM) ｶﾗ (TO) ﾏﾃﾞ ｦ ﾋｮｳｼﾞ
1950:                   ;
1960:                   ;
1970:   D367            DUMP:
1980:   D367 ED5B1FD4            LD      DE,(FROM)
1990:   D36B 2A21D4              LD      HL,(TO) ;(FROM) ｶﾗ (TO) ﾊ
2000:                                            ; (DE) ｶﾗ (HL) ﾄｽﾙ
2010:   D36E E7                  RST     20H     ;DCOMPR[BIOS] , (DE) > (HL) ?
```

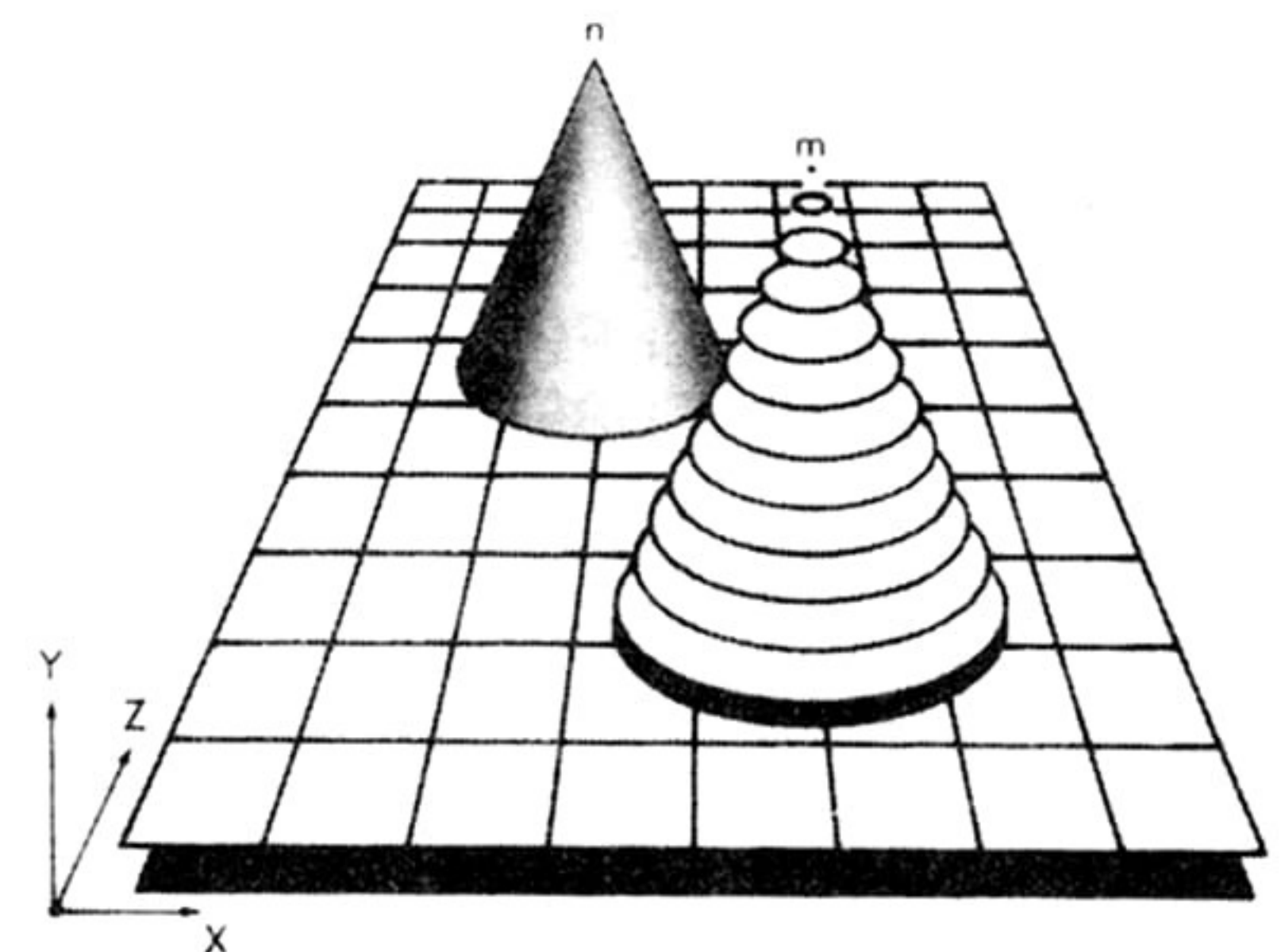
```
2020:   D36F 3854                JR      C,ERR    ;YES,ERROR
2030:                    ;
2040:   D371 E5                  PUSH    HL
2050:   D372 D5                  PUSH    DE
2060:   D373 C5                  PUSH    BC       ;レジ゛スタ セーブ゛
2070:   D374             DUMP2:
2080:   D374 CD93D3              CALL    ADRP     ;ADDR. ブ゛ブ゛ン ヲ ヒョウジ゛
2090:   D377 0608                LD      B,8      ;1ギ゛ョウ 8 バ゛イト ブ゛ン ノ ループ゜ カウンタ
2100:   D379             ONELIN:
2110:   D379 CDB1D3              CALL    DATAP    ;デ゛ータ ブ゛ブ゛ン ヲ ヒョウジ゛
2120:   D37C 13                  INC     DE       ;ツギ゛ ノ 1 バ゛イト
2130:   D37D E7                  RST     20H      ;DCOMPR , (DE) > (HL) ?
2140:   D37E 380D                JR      C,EXIT   ;YES,シュウリョウ
2150:   D380 10F7                DJNZ    ONELIN   ;1ギ゛ョウ 8バ゛イト ブ゛ン クリカエス
2160:                    ;
2170:   D382 CDDED3              CALL    CRLF     ;1ギ゛ョウ ブ゛ン シュウリョウ ナノデ゛
2180:                                             ; カイギ゛ョウ スル
2190:   D385 CDB700              CALL    BREAKX   ;CTRL-STOP キー ノ チェック
2200:   D388 DA00EC              JP      C,MON    ;オサレテ イレバ゛ MSX-MONITOR ニ トブ゛
2210:   D38B 18E7                JR      DUMP2    ;オサレテ イナイ ノデ゛ ツギ゛ ノ ギ゛ョウ ヲ
2220:                                             ; ヒョウジ゛ スル
2230:                    ;
2240:   D38D             EXIT:
2250:   D38D C1                  POP     BC
2260:   D38E D1                  POP     DE
2270:   D38F E1                  POP     HL       ;レジ゛スタ ヲ フッキ
2280:   D390 C300D3              JP      KEYIN    ;フタタビ゛ KEYIN ヘ
2290:                    ;
2300:                    ;
2310:                    ;
2320:                    ; コレヨリ サブ゛ルーチン グ゛ン
2330:                    ;
2340:                    ;
2350:                    ;
2360:                    ; ADDR. ブ゛ブ゛ン ヒョウジ゛ サブ゛ルーチン
2370:                    ;
2380:   D393             ADRP:
2390:   D393 7A                  LD      A,D      ;Dレジ゛スタ ハ ADDR.(16シンスウ4ケタ) ノ
2400:                                             ; ジ゛ョウイ 4ケタ
2410:   D394 CDFDD3              CALL    TOASC    ;ASCII コード゛ デ゛ ヒョウゲ゛ン サレタ
2420:                                             ; 2モジ゛ ニ ヘンカン,(BC) ニ
2430:   D397 78                  LD      A,B
2440:   D398 CDA200              CALL    CHPUT    ;ADDR.4ケタ ノ ウチ,イチバ゛ン ウエ ノ
2450:                                             ; ケタ ヲ ヒョウジ゛
2460:   D39B 79                  LD      A,C
2470:   D39C CDA200              CALL    CHPUT    ;ADDR.4ケタ ノ ウチ,2ケタ メ ヲ ヒョウジ゛
2480:   D39F 7B                  LD      A,E      ;Eレジ゛スタ ハ ADDR. ノ カイ 2ケタ
2490:   D3A0 CDFDD3              CALL    TOASC    ;ASCII 2モジ゛ ニ ヘンカン,(BC) ニ イレル
2500:   D3A3 78                  LD      A,B
2510:   D3A4 CDA200              CALL    CHPUT    ;ADDR.4ケタ ノ ウチ,3ケタ メ ヲ ヒョウジ゛
2520:   D3A7 79                  LD      A,C
2530:   D3A8 CDA200              CALL    CHPUT    ;ADDR.4ケタ ノ ウチ,4ケタ メ ヲ ヒョウジ゛
2540:   D3AB 3E20                LD      A,' '    ;ADDR. ブ゛ ト DATA ブ゛ トノ アイダ゛ ニ
2550:                                             ; 1ツ ノ SPACE
2560:   D3AD CDA200              CALL    CHPUT
2570:   D3B0 C9                  RET
2580:                    ;
2590:   D3B1             DATAP:
2600:   D3B1 C5                  PUSH    BC       ;DJNZ ノ カウンタ ヲ タイヒ
2610:   D3B2 1A                  LD      A,(DE)   ;メサ゛ス ADDR.ノ DATAヲ (DE) カラ Aヘ
2620:   D3B3 CDFDD3              CALL    TOASC    ;ASCII コード゛ 2モジ゛ ニ ヘンカン,
2630:                                             ; (BC) ニ イレル
2640:   D3B6 78                  LD      A,B
2650:   D3B7 CDA200              CALL    CHPUT    ;DATA(2モジ゛)ノ ウチ,1モジ゛メ ヲ ヒョウジ゛
2660:   D3BA 79                  LD      A,C
2670:   D3BB CDA200              CALL    CHPUT    ;DATA ノ 2モジ゛ メ ヲ ヒョウジ゛
2680:   D3BE 3E20                LD      A,' '
```

```
2690:   D3C0 CDA200            CALL   CHPUT  ;DATA カン ノ SPACE ヲ ヒョウジ"
2700:   D3C3 C1                POP    BC     ;DJNZ ノ カウンタ ヲ フッキ
2710:   D3C4 C9                RET
2720:                  ;
2730:   D3C5           ERR:
2740:   D3C5 3E3F              LD     A,'?'
2750:   D3C7 CDA200            CALL   CHPUT  ;'?' ヲ 1モジ" ヒョウジ"
2760:   D3CA C300D3            JP     KEYIN  ;KEYIN ヲ ヤリナオシ.
2770:                  ;
2780:   D3CD           TOHEX:
2790:   D3CD F5                PUSH   AF     ;モジ" DATA ヲ ニガ"ス
2800:   D3CE FE3A              CP     3AH    ;'0' カラ '9' マデ" ノ モジ" カ ?
2810:   D3D0 3802              JR     C,SUUJI ;YES
2820:   D3D2 D607              SUB    7      ;'A' カラ 'F' マデ"ナラ 7 ヲ ヒイテ
2830:                                        ; *AH カラ *FH ニ ナオス
2840:   D3D4           SUUJI:
2850:   D3D4 E60F              AND    0FH    ;カイ 4ビ"ット ヲ ノコシテ マスク スル
2860:   D3D6 29                ADD    HL,HL  ;HL*2
2870:   D3D7 29                ADD    HL,HL  ;HL*4
2880:   D3D8 29                ADD    HL,HL  ;HL*8
2890:   D3D9 29                ADD    HL,HL  ;HL*16,16バ"イ ハ 16シン 1ケタブ"ン ヲ
2900:                                        ; ヒダ"リ シフト シタノト オナジ"
2910:   D3DA B5                OR     L      ;Lハ *0ナノデ",ORデ" Aノ ジ"ョウイ4ビ"ット
2920:                                        ; ニ Lノ *カ" ウメコマレル
2930:   D3DB 6F                LD     L,A    ;Lニ ソノ アタイ ヲ イレル
2940:   D3DC F1                POP    AF     ;モジ" DATA ヲ フッキ
2950:   D3DD C9                RET
2960:                  ;
2970:   D3DE           CRLF:
2980:   D3DE 3E0D              LD     A,CR   ;CR コード"
2990:   D3E0 CDA200            CALL   CHPUT
3000:   D3E3 3E0A              LD     A,LF   ;LF(カイギ"ョウ) コード"
3010:   D3E5 CDA200            CALL   CHPUT
3020:   D3E8 C9                RET
3030:
3040:   D3E9           CHKHEX:
3050:   D3E9 FE30              CP     '0'        ;'0'イジ"ョウ カ ?
3060:   D3EB 380C              JR     C,NOTHEX ;NO,16シンスウモジ" ジ"ャナイ
3070:   D3ED FE3A              CP     '9'+1      ;'9'イカ カ ?
3080:   D3EF 380A              JR     C,YESHEX ;YES,'0' カラ '9' ノ 16シンスウ モジ"
3090:   D3F1 FE41              CP     'A'        ;'A'イジ"ョウ カ ?
3100:   D3F3 3804              JR     C,NOTHEX ;NO
3110:   D3F5 FE47              CP     'F'+1      ;'F'イカ カ ?
3120:   D3F7 3802              JR     C,YESHEX ;YES,'A' カラ 'F' ノ 16シンスウ モジ"
3130:   D3F9           NOTHEX:
3140:   D3F9 37                SCF               ;16シンスウモジ" ジ"ャナイ トキ,
3150:                                            ; Cフラグ" ヲ ON シテ RET
3160:   D3FA C9                RET
3170:                  ;
3180:   D3FB           YESHEX:
3190:   D3FB B7                OR     A          ;16シンスウモジ" ダ"ッタ トキ,
3200:                                            ; Cフラグ" ヲ OFF シテ RET
3210:   D3FC C9                RET
3220:                  ;
3230:   D3FD           TOASC:
3240:   D3FD F5                PUSH   AF     ;モト ノ DATA ヲ ニガ"ス
3250:   D3FE E6F0              AND    0F0H   ;ジ"ョウイ 4ビ"ット ヲ ノコシテ マスク
3260:   D400 CD0ED4            CALL   ASC2   ;ジ"ョウイ 4ビ"ット ヨウ ノ
3270:                                        ; ASCII コード" ヘンカン ルーチン
3280:   D403 47                LD     B,A    ;Bレジ"スタ ニ シマウ
3290:   D404 F1                POP    AF     ;DATA ヲ オモイダ"ス
3300:   D405 F5                PUSH   AF     ;フタタビ" DATA ヲ ニガ"ス
3310:   D406 E60F              AND    0FH    ;ツギ"ハ カイ 4ビ"ット ヲ ノコシテ マスク
3320:   D408 CD16D4            CALL   ASC3   ;カイ 4 ビ"ット ヨウ ノ
3330:                                        ; ASCII コード" ヘンカン ルーチン
3340:   D40B 4F                LD     C,A    ;Cレジ"スタ ニ シマウ
3350:   D40C F1                POP    AF     ;DATA ヲ フッキ
```

```
3360:   D40D C9                     RET
3370:                 ;
3380:   D40E          ASC2:
3390:   D40E CB3F                   SRL     A       ;ｼﾞｮｳｲ 4ﾋﾞｯﾄ ｦ ｶｲ 4ﾋﾞｯﾄ ﾆ ｼﾌﾄ
3400:   D410 CB3F                   SRL     A       ;ﾐｷﾞ ｼﾌﾄ,ｱｲﾀ 1ﾋﾞｯﾄ ﾆﾊ 0 ｶﾞ ﾊｲﾙ
3410:   D412 CB3F                   SRL     A
3420:   D414 CB3F                   SRL     A       ;4ｶｲ ｼﾌﾄ ｽﾚﾊﾞ ｼﾞｮｳｲ 4ﾋﾞｯﾄ ﾊ
3430:                                               ; ｶｲ ﾆ ｲﾄﾞｳ ｽﾙ
3440:   D416          ASC3:
3450:   D416 FE0A                   CP      0AH     ;ｶｲ 4ﾋﾞｯﾄ ﾊ A ｲｶ ｶ ?
3460:   D418 3802                   JR      C,SUUJI2 ;YES,00 ｶﾗ 09
3470:   D41A C607                   ADD     A,7H    ;0A ｶﾗ 0F ﾀﾞｯﾀﾗ 37 ｦ ﾀｽｺﾄ ﾆ ﾅﾙ
3480:   D41C          SUUJI2:
3490:   D41C C630                   ADD     A,30H   ;30H,ﾏﾀﾊ37H ｦ ﾀｽﾄ
3500:                                               ; ´0´ ｶﾗ ´F´ ﾏﾃﾞ ﾉ ｺｰﾄﾞ ﾆ ﾅﾙ
3510:   D41E C9                     RET
3520:                 ;
3530:                 ;
3540:                 ;
3550:                 ;
3560:   D41F 0000     FROM:         DEFW    0000H ;DUMP ｶｲｼ ADDR. ｦ ｼﾏｳ ﾊﾞｼｮ
3570:   D421 0000     TO:           DEFW    0000H ;DUMP ｼｭｳﾘｮｳ ADDR. ｦ ｼﾏｳ ﾊﾞｼｮ
```

基本テクニックをまとめてみよう

5 アセンブル

ニモニック（擬似命令を含む）をアセンブラにかけてオブジェクトにします。このときエラーが出たら修正せねばなりません。アセンブルのパス1，2の両方でエラーが出なくなるまで修正を重ねます。リスト3-1は，アセンブルしたときの出力（アセンブル・リスト）です。

6 基本テクニックをまとめてみよう

デバッグ

実際に動作させてチェックします。起こりうるすべての場合をチェックすることは不可能ですが，考えつく限りの動作をさせてみましょう。アセンブルでエラーにならなくとも，論理的なミスが含まれているために設計どおりの動作をしないことは多いものです。暴走などしたら最悪ですね。

メモリダンプ・ルーチンの場合，先に挙げた5通りのコマンド入力しか許していませんので，これら以外はみなエラーとして処理されるはずです。また，開始・終了アドレスが4桁以外の場合や16進数文字以外の文字を含んでいる場合，および開始>終了アドレスとなっている場合も試してみる必要があります。開始＝終了という関係のとき正確に1バイトだけ出力されるかということも重要なチェック項目のひとつです。

これらのチェックにより異常が発見された場合，原因を究明して取り除かねばなりません。ことによってはフローチャートから変更する必要が出てくるでしょう。

以上のような手順を踏んで「メモリダンプ・ルーチン」はできあがりました。まだムダなことをしている場所もあります。バグもあるかもしれません。みなさんで改造してください。チェックサムをつける，プリンタに出力する，などはそれほど難しいことではありませんから。

また，このメモリダンプ・ルーチンを解読して，他のプログラムにも挑戦してみましょう。

動作例1　“7000，707F ↵”と入力したとき

```
7000 00 E1 C9 7D CD DE 72 7C
7008 C3 DE 72 CD D4 72 6F CD
7010 D4 72 67 C9 0E D0 CD B8
7018 70 CD E9 72 C1 CD 0B 70
7020 09 EB CD 0B 70 09 E5 CD
7028 0B 70 22 BF FC EB D1 CD
7030 D4 72 77 E7 28 03 23 18
7038 F6 CD E7 00 C3 F4 6E D6
7040 91 28 02 AF 01 2F 23 FE
```

```
7048 01 F5 CD 8C 70 0E D3 CD
7050 B8 70 F1 32 F8 F7 DC 87
7058 62 3A F8 F7 FE 01 32 F5
7060 F3 F5 CD EA 54 F1 2A 76
7068 F6 CD 5D 71 20 10 22 C2
7070 F6 21 D7 3F CD 78 66 2A
7078 76 F6 E5 C3 37 42 23 EB
```

動作例2　1につづいて“，↵”と入力したとき

```
7080 2A C2 F6 E7 DA 71 70 1E
7088 14 C3 6F 40 2B D7 20 08
7090 E5 21 66 F8 06 06 18 19
7098 CD 64 4C E5 CD 0F 68 2B
70A0 2B 46 0E 06 21 66 F8 1A
70A8 77 23 13 0D 28 08 10 F7
70B0 41 36 20 23 10 FB E1 C9
70B8 CD E9 72 06 0A CD D4 72
70C0 B9 20 F5 10 F8 21 71 F8
70C8 E5 06 06 CD D4 72 77 23
70D0 10 F9 E1 11 66 F8 06 06
70D8 1A 13 FE 20 20 04 10 F8
70E0 18 0D 11 66 F8 06 06 1A
70E8 BE 20 0A 23 13 10 F8 21
70F0 FF 70 C3 0D 71 C5 21 06
70F8 71 CD 0D 71 C1 18 B9 46
```

動作例3　2につづいて“，712F ↵”と入力したとき

```
7100 6F 75 6E 64 3A 00 53 6B
7108 69 70 20 3A 00 ED 5B 1C
7110 F4 13 7A B3 C0 CD 78 66
7118 21 71 F8 06 06 7E 23 DF
7120 10 FB C3 28 73 CD F8 72
7128 06 0A CD DE 72 10 FB 06
```

4章 ものにしよう 実践テクニック

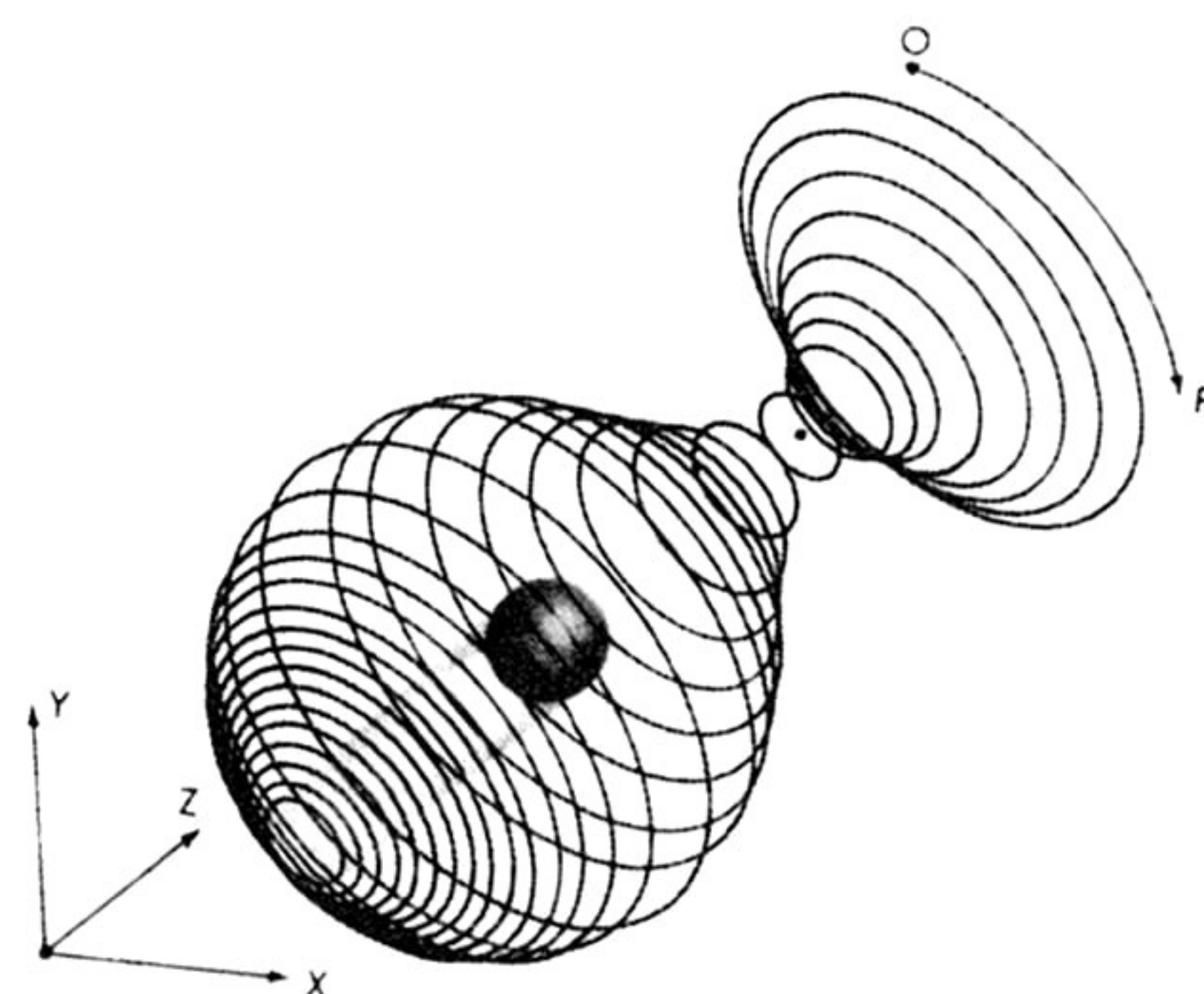
マシン語の実践テクニックを自分のものにするための，ちょっと高級なルーチン集。マシン語で本格的な計算をするには？　マシン語で配列なんか使えるんだろうか。ルーチン間でデータを渡すには？などなど，これさえ知ればマシン語のプログラムは大丈夫。

前章までを読んで，「マシン語は難しくない。ただ複雑なだけだ」ということがわかっていただけたでしょうか？　そして，その複雑さも，実はレジスタに数値を入れたり，足し算をしたり，とても単純なことが組み合わさっているだけなのです。

その組み合わせを自由自在に作ることができれば，あなたは「マシン語がわかる」人になるわけです。この章ではいろいろな組み合わせで，配列のデータをとり扱う方法と，かけ算，割り算のサブルーチンについて話を進めます。

この章のアセンブル・リストの多くは，あなたが自分でプログラムを作るときに利用できるようになっています。そのため，あるプログラムの動作を実際に確認するときには以下の注意が必要です。

①アセンブラを実行直後CLEAR　100，&HD300↵
　としてください。
②結果がレジスタに残るものを直接見たいときは，ＲＥＴ命令のところにブレーク・ポイントをセットして実行します。
③　アセンブル・リストでは１行に４バイトまでしか16進（オブジェクト）を表示しません。モニタで直接ダンプ・リストを入力するときは注意してください。
④　あるサブルーチンが他のサブルーチンをコールするときは，当然対象サブルーチンがメモリ中に存在することが必要です。

1 ものにしよう実践テクニック

パラメータの引き渡し方法

1. アドレス渡し

これまでは，サブルーチンにパラメータを渡すのにレジスタを使いました。では，次のようなときにはどうしたらよいでしょう。

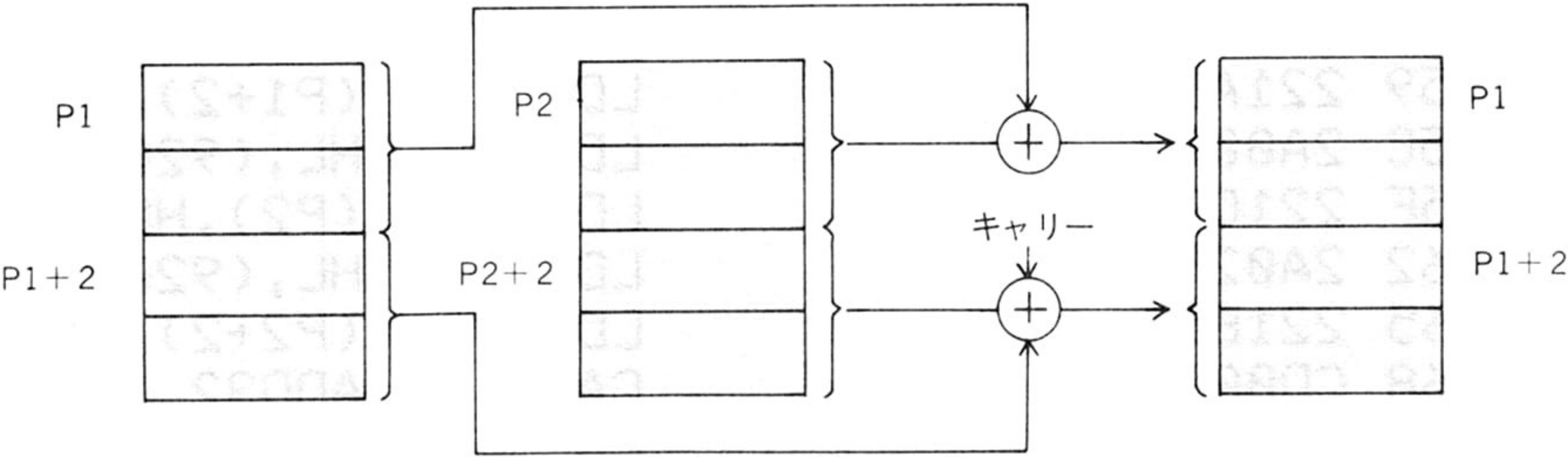
32ビットの整数どうしを足すサブルーチンをつくりたい

Z80のレジスタはペアにしても16ビットまでしか入りませんから，HLやDEに値を入れるというわけにはいきません。

パラメータを置くメモリを決めてつくると**リスト4-1**のようなサブルーチンになります。このサブルーチンはP1番地からの2バイトとP2番地からの2バイトを足し，P1＋2番地からの2バイトとP2＋2番地からの2バイトとキャリーとを足します(図4-1)。

リスト4-1

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                    ORG     0D400H
110:    D400 2A18D4    ADD32:   LD      HL,(P1)
120:    D403 ED5B1CD4           LD      DE,(P2)
130:    D407 19                 ADD     HL,DE
140:    D408 2218D4             LD      (P1),HL
150:    D40B 2A1AD4             LD      HL,(P1+2)
160:    D40E ED5B1ED4           LD      DE,(P2+2)
170:    D412 ED5A               ADC     HL,DE
180:    D414 221AD4             LD      (P1+2),HL
190:    D417 C9                 RET
200:                   ;
210:    D418           P1:      DEFS    4
220:    D41C           P2:      DEFS    4
```

図4-1

結果はP1番地からの4バイトに入ります。

このサブルーチンを使って，9000H番地からの4バイトと9200H番地からの4バイトを足し，結果を 9000H 番地に置くプログラムをつくると**リスト4-2**のようになります。このプログラムを実行するためには，D400H番地から**リスト 4-1** のプログラムが格納されていなくてはなりません。実行はD450H番地からです。このプログラムは，パラメータを渡すのにかなり手数が必要です。こういうときはパラメータを渡すサブルーチンをつくってしまいましょう。**リスト 4-3** はHLで示すアドレスからの4バイトをP1番地からの4バイトに転送します。

同様に**リスト 4-4** はHLの示すアドレスからの4バイトをＰ２番地からの4バイトに，**リスト4-5** はＰ２番地からの4バイトをＤＥの示すアドレスからの4バイトに転送します。

これらのサブルーチンを使って，**リスト 4-2**と同じ働きをするプログラムを作ると P106 の**リスト 4-6**のようになります。このプログラムを実行するためには，**リスト 4-1，4-3，4-4，4-5**のプログラムが必要です。実行は D480H 番地からとなります。

このプログラムでは，HLレジスタやＤＥレジスタにアドレスを入れて，サブルーチンをコールしています。このように，パラメータとして値そのものでなく，パラメータの置いてあるアドレスを渡す方法を「アドレス渡し」と呼びます。

リスト4-2

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400 =          ADD32     EQU      0D400H
110:    D418 =          P1        EQU      0D418H
120:    D41C =          P2        EQU      0D41CH
130:
140:    D450                      ORG      0D450H
150:    D450 2A0090               LD       HL,(9000H)
160:    D453 2218D4               LD       (P1),HL
170:    D456 2A0290               LD       HL,(9002H)
180:    D459 221AD4               LD       (P1+2),HL
190:    D45C 2A0092               LD       HL,(9200H)
200:    D45F 221CD4               LD       (P2),HL
210:    D462 2A0292               LD       HL,(9202H)
220:    D465 221ED4               LD       (P2+2),HL
230:    D468 CD00D4               CALL     ADD32
240:    D46B 2A18D4               LD       HL,(P1)
```

```
250:    D46E 220090             LD      (9000H),HL
260:    D471 2A1AD4             LD      HL,(P1+2)
270:    D474 220290             LD      (9002H),HL
280:    D477 C9                 RET
```

リスト4-3～5

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D418 =          P1      EQU     0D418H
110:    D41C =          P2      EQU     0D41CH
120:
130:    D450                    ORG     0D450H ┐
140:    D450 1118D4     TOP1:   LD      DE,P1  │
150:    D453 010400             LD      BC,4   │ 4-3
160:    D456 EDB0               LDIR           │
170:    D458 C9                 RET            ┘
180:                                           ┐
190:    D459 111CD4     TOP2:   LD      DE,P2  │
200:    D45C 010400             LD      BC,4   │ 4-4
210:    D45F EDB0               LDIR           │
220:    D461 C9                 RET            ┘
230:                                           ┐
240:    D462 2118D4     FROMP1: LD      HL,P1  │
250:    D465 010400             LD      BC,4   │ 4-5
260:    D468 EDB0               LDIR           │
270:    D46A C9                 RET            ┘
```

この方法を使って，32ビットの足し算をする別のサブルーチンをつくると次の頁の**リスト4-7**のようになります。このサブルーチンは，まずHLの示すアドレスの内容とDEの示すアドレスの内容を足します（**図4-2**）。次に，HLとDEを1だけ増します（**図4-3**）。これを4回繰り返すためにBレジスタに4を入れ，DJNZ命令を使います。

足し算のときは，下位のバイトからの桁上りを考えてADC命令を使います。最初のバイトの足し算ではキャリーは0でないとまずいので，OR A命令を使っています。INC命令とDJNZ命令ではキャリーは変化しないことも覚えておいてください。

次頁の**リスト4-7**を使って**リスト4-2**と同じ働きをするプログラムをつくるとP107の**リスト4-8**のようになります。このプログラムを実行するためには，**リスト4-7**のプログラムが必要です。実行

はD450H番地からです。**リスト 4-2**と比較すると，ずいぶんすっきりしたプログラムになりました。

リスト 4-6

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    D400 =        ADD32      EQU      0D400H
110:    D450 =        TOP1       EQU      0D450H
120:    D459 =        TOP2       EQU      0D459H
130:    D462 =        FROMP1     EQU      0D462H
140:                  ;
150:    D480                     ORG      0D480H
160:    D480 210090              LD       HL,9000H
170:    D483 CD50D4              CALL     TOP1
180:    D486 210092              LD       HL,9200H
190:    D489 CD59D4              CALL     TOP2
200:    D48C CD00D4              CALL     ADD32
210:    D48F 110090              LD       DE,9000H
220:    D492 CD62D4              CALL     FROMP1
230:    D495 C9                  RET
```

リスト 4-7

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    D400                     ORG      0D400H
110:    D400 0604     ADD32:     LD       B,4
120:    D402 B7                  OR       A
130:    D403 1A       LOOP:      LD       A,(DE)
140:    D404 13                  INC      DE
150:    D405 8E                  ADC      A,(HL)
160:    D406 77                  LD       (HL),A
170:    D407 23                  INC      HL
180:    D408 10F9                DJNZ     LOOP
190:    D40A C9                  RET
```

図4-2

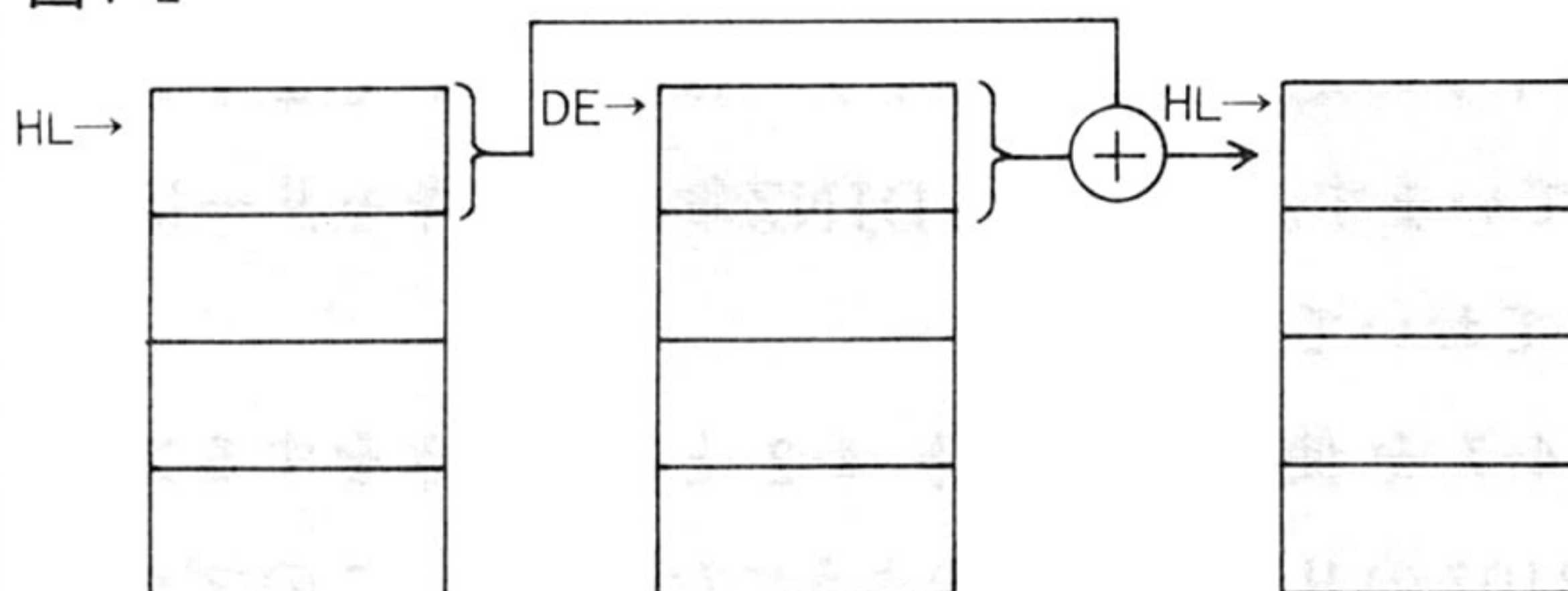
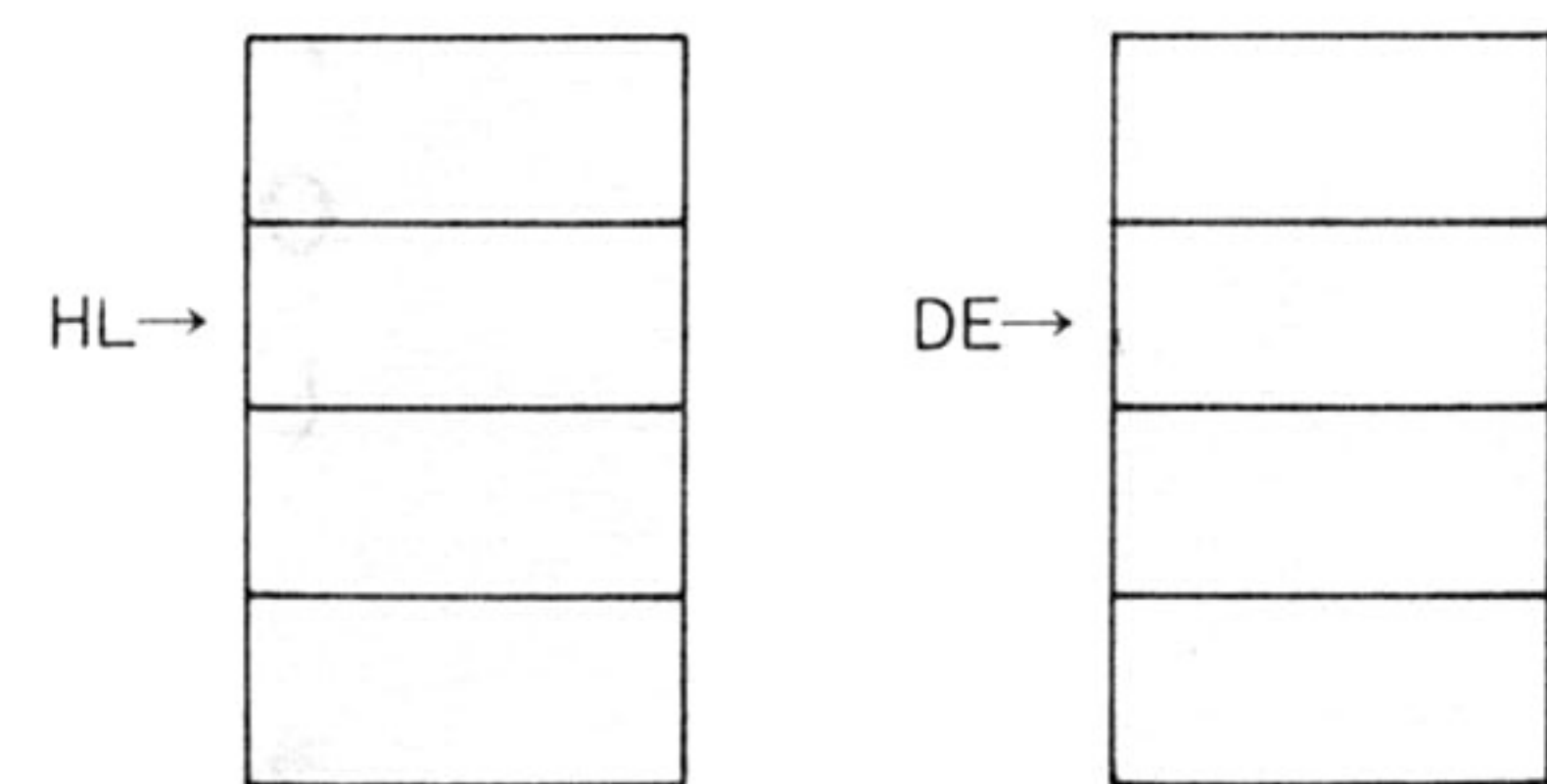



図4-3

リスト4-8

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400 =        ADD32       EQU      0D400H
110:    D450                      ORG      0D450H
120:    D450 210090               LD       HL,9000H
130:    D453 110092               LD       DE,9200H
140:    D456 CD00D4               CALL     ADD32
150:    D459 C9                   RET
```

2.インライン・パラメータ

次のようなときもアドレス渡しを活用します。

文字列を画面に出力する

HLで文字列の先頭アドレスを示し，文字列の最後を 0とすると（**図4-4**），P108の**リスト4-9**のようなサブルーチンができます。

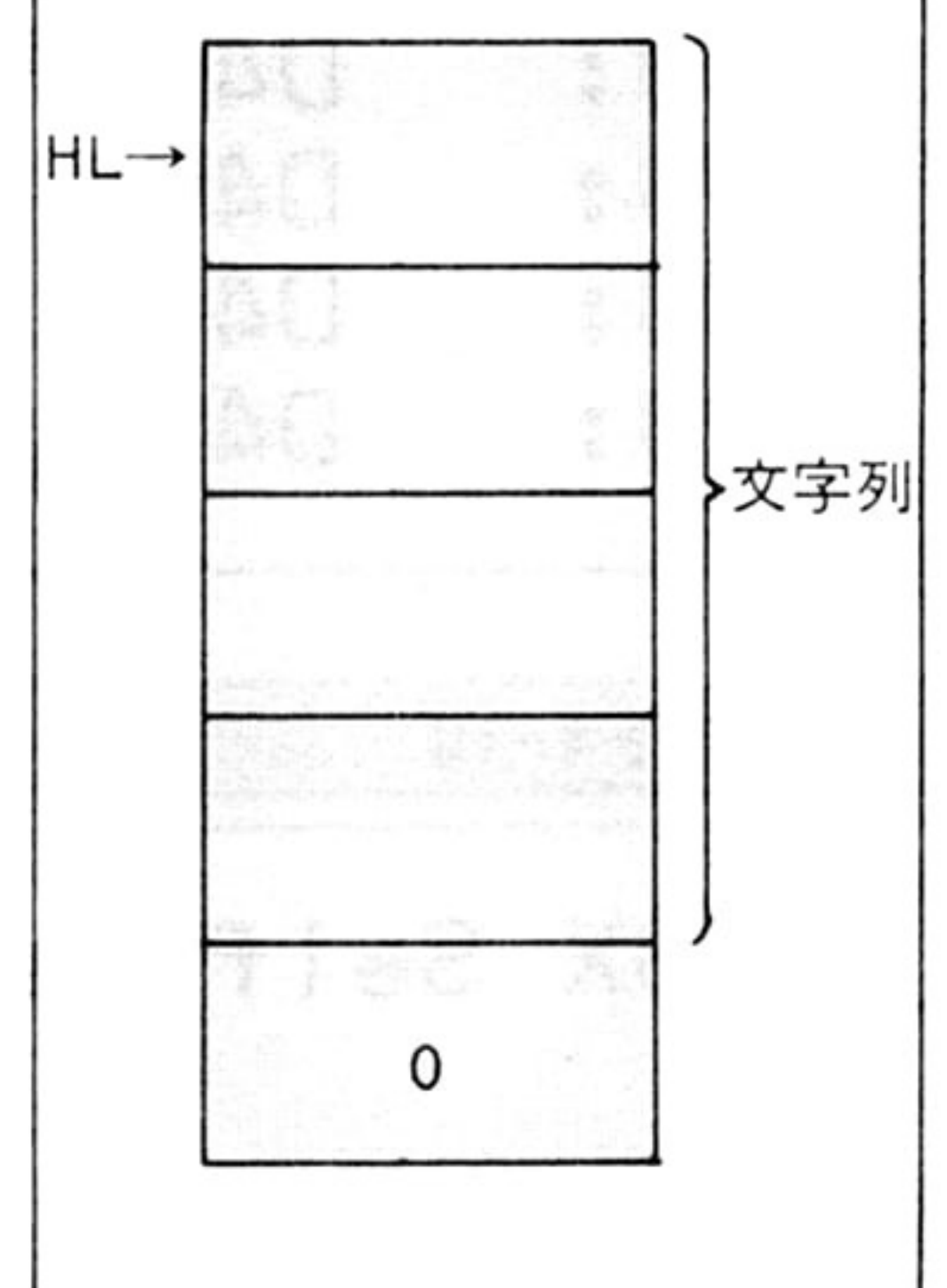
このサブルーチンを使って文字列を表示するのが，P108の **リスト 4-10** のプログラムです。このプログラムを実行するためには**リスト 4-9** のプログラムが必要です。実行はD450H番地からです。

図4-4

ここで，レジスタを使わないでサブルーチンに文字列の先頭アドレスを渡す方法があります。それはP109の**リスト 4-11** のように,CALL命令のすぐ後に文字列を置く方法です。

これを処理するサブルーチンはP109の **リスト 4-12** のようになります。**リスト 4-11** のプログラムを実行するには **リスト 4-12** のプログラムが必要です。実行はD450H番地からです。

リスト4-11 でCALL命令が実行されると，戻り番地としてスタックにMSGのアドレスが置かれます(**図 4-5**)。**リスト4-12** では最初にPOP HL命令を実行しますから，HLレジスタにMSGのアドレスが置かれることになります。あとは，0がくるまで **リスト 4-9** と同じことを繰り返します。

Aレジスタの値が0になったとき，HLは文字列の最後のアドレスの次のアドレスを示しています(**図4-6**)。

そこで，JP (HL)命令を使ってHLの示すアドレスにジャンプすればよいのです。

普通のサブルーチンのようにRET命令を使ってメイン・ルーチンに戻ろうとしてはいけません。スタックに戻り番地が置いてありませんから。

このように,CALL命令のすぐ後に置いたパラメータをインライン・パラメータと呼びます。

リスト4-9

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    00A2 =         CHPUT         EQU      00A2H
110:
120:    D400                         ORG      0D400H
130:    D400 7E        LOUT:         LD       A,(HL)
140:    D401 23                      INC      HL
150:    D402 B7                      OR       A
160:    D403 2805                    JR       Z,EXIT
170:    D405 CDA200                  CALL     CHPUT
180:    D408 18F6                    JR       LOUT
190:    D40A C9        EXIT:         RET
```

リスト4-10

```
 MSX Self Assembler  Rev 1.0     PAGE  1


100:    D400 =         LOUT          EQU  0D400H
110:
120:    D450                         ORG  0D450H
130:    D450 2157D4                  LD   HL,MSG
140:    D453 CD00D4                  CALL LOUT
150:    D456 C9                      RET
160:
170:    D457 4920616D MSG:           DEFM ´I am a computer.´
180:    D467 0D0A00                  DEFB 0DH,0AH,0
190:    D46A C9        EXIT:         RET
```

リスト4-11

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    D400 =          OUTL          EQU      0D400H
110:
120:    D450                          ORG      0D450H
130:    D450 CD00D4                   CALL     OUTL
140:    D453 486F7720                 DEFM     'How are you?'
150:    D45F 0D0A00                   DEFB     0DH,0AH,0
160:    D462 C9                       RET
```

リスト4-12

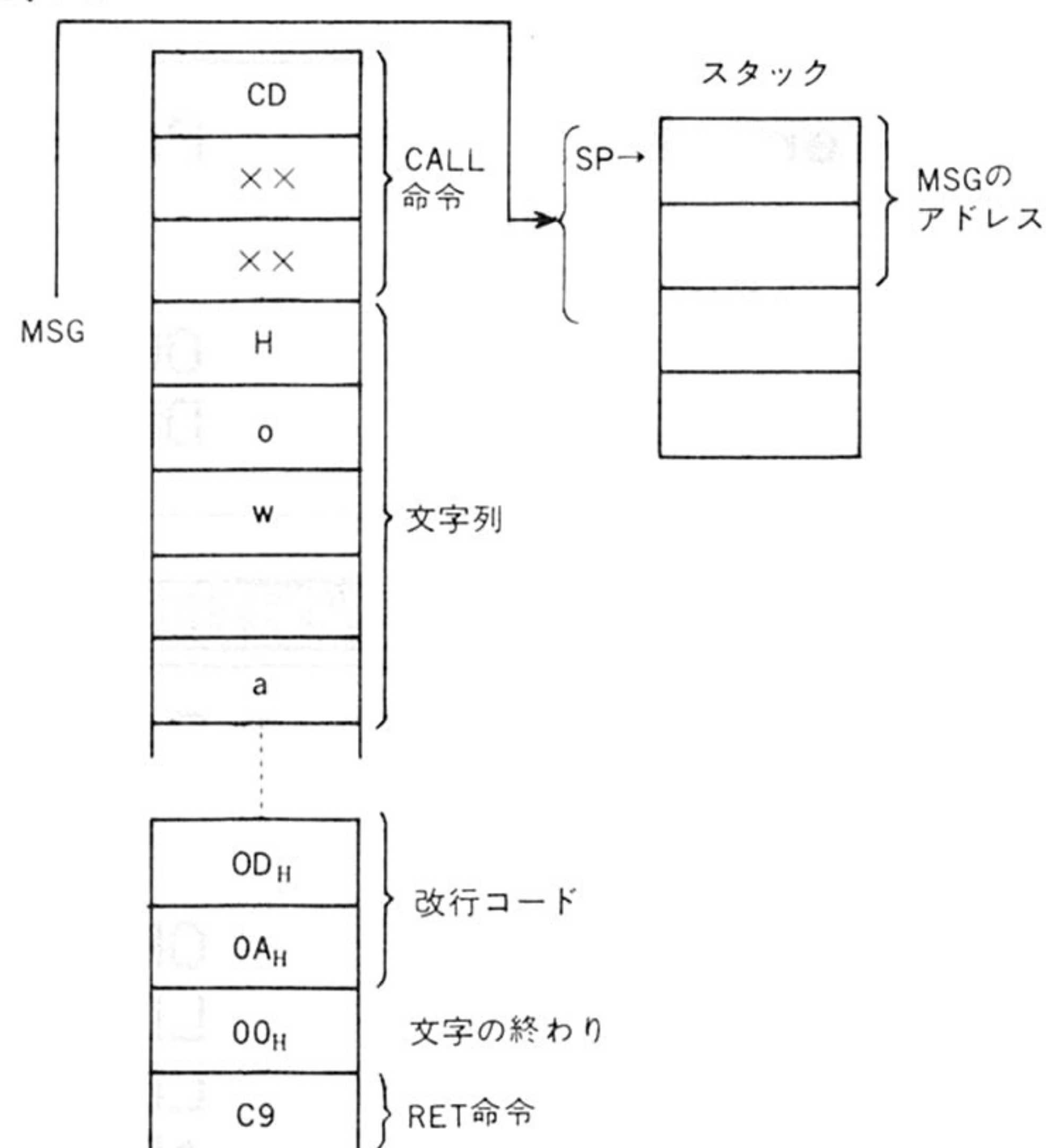
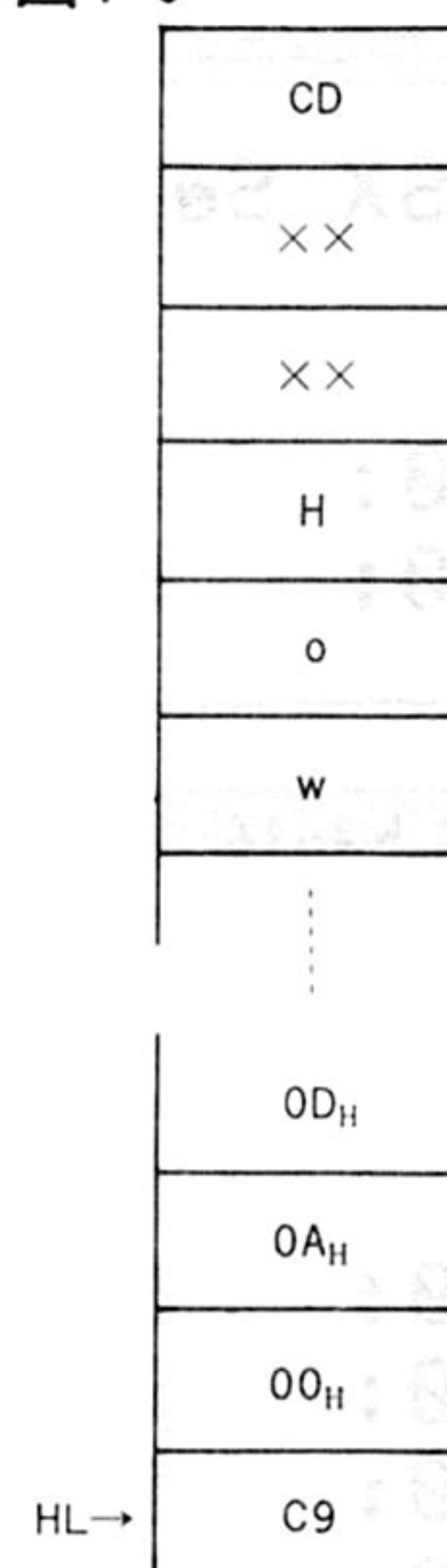
```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    00A2 =          CHPUT         EQU      00A2H
110:
120:    D400                          ORG      0D400H
130:    D400 E1         OUTL:         POP      HL
140:    D401 7E         LOOP:         LD       A,(HL)
150:    D402 23                       INC      HL
160:    D403 B7                       OR       A
170:    D404 2805                     JR       Z,EXIT
180:    D406 CDA200                   CALL     CHPUT
190:    D409 18F6                     JR       LOOP
200:    D40B E9         EXIT:         JP       (HL)
```

図4-5

CD
××
××
CALL命令
MSG
H
o
w
a
文字列
$0D_H$
$0A_H$
改行コード
00_H
文字の終わり
C9
RET命令
スタック
SP→
MSGのアドレス

図4-6

CD
××
××
H
o
w
$0D_H$
$0A_H$
00_H
HL→
C9

もの にしよう実践テクニック

2 マシン語で配列を使う

1.1次元配列

BASICではDIM TBL(10)などとすることにより,配列変数を使うことができます。ここではマシン語で同じような機能を実現してみましょう。

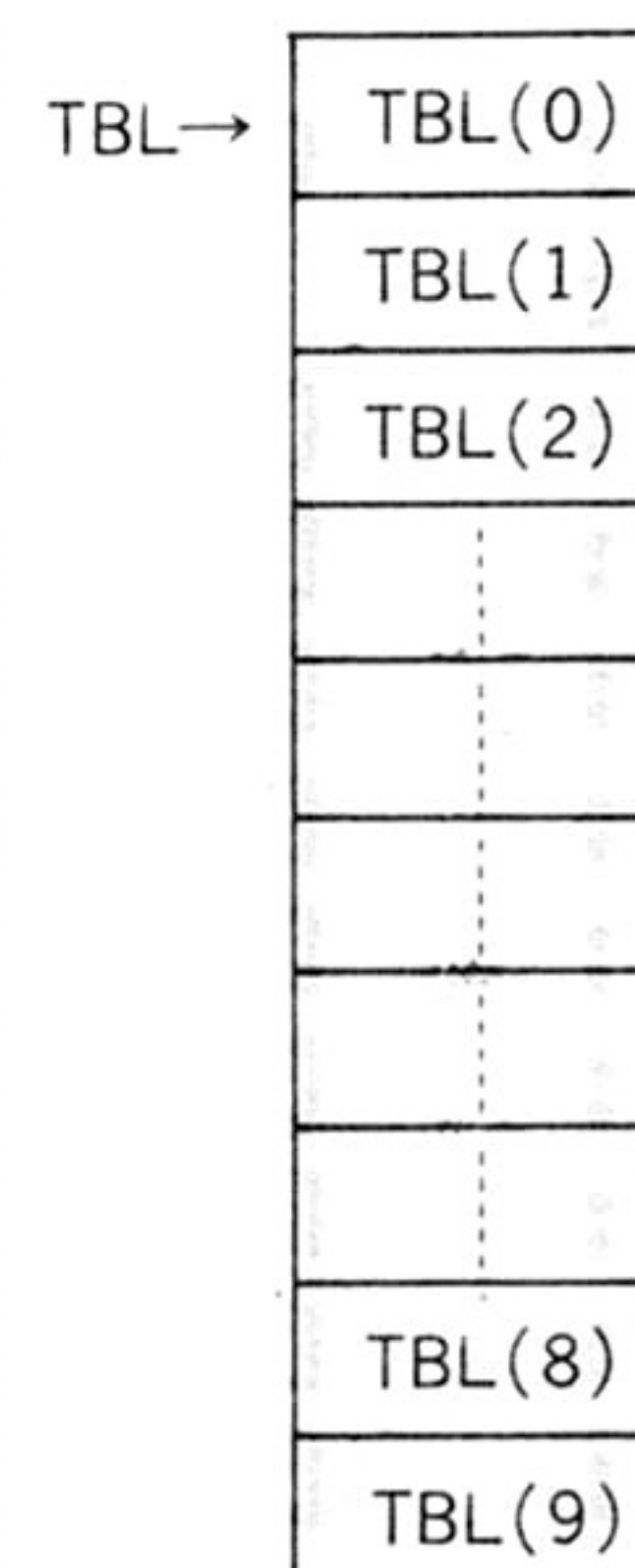
配列のひとつひとつ，すなわちTBL(0), TBL(1), …を「配列の要素」といいます。まず，要素の大きさが1バイトのときを考えます。

要素の数が10個の配列を使うことを宣言する

これは **リスト 4-13** のようにします。これは **図 4-7** のようにメモリを利用するという意味で，DEFSという命令を実行するという意味ではありませんから注意が必要です。

TBL(5)の値をAレジスタに入れるプログラムは **リスト4-14** のようになります。これは配列の先頭アドレスをHLレジスタに，要素の番号をBCレジスタにいれて足し算をしています。これにより，HLレジスタは求めたい配列の要素の置いてあるアドレスを示すようになります。

図4-7

TBL→	TBL(0)
	TBL(1)
	TBL(2)
	⋮
	⋮
	⋮
	⋮
	⋮
	TBL(8)
	TBL(9)

リスト4-13

```
 MSX Self Assembler   Rev 1.0     PAGE      1


100:    D400                      ORG       0D400H
110:    D400            TBL:      DEFS      10
```

リスト4-14

```
 MSX Self Assembler   Rev 1.0     PAGE      1


100:    D400                      ORG       0D400H
110:    D400 2109D4               LD        HL,TBL
120:    D403 010500               LD        BC,5
130:    D406 09                   ADD       HL,BC
```

```
140:    D407 7E                LD      A,(HL)
150:    D408 C9                RET
160:
170:    D409          TBL:     DEFS    10
```

逆にAレジスタの値をTBL(5)に入れるには，**リスト 4-15** のようにします。

ここで注意しなければならないのは，要素の番号を誤って大きくしすぎたときです。

たとえば，**リスト 4-16** のプログラムは配列の範囲以外のメモリの内容を破壊してしまいます(**図 4-8**)。

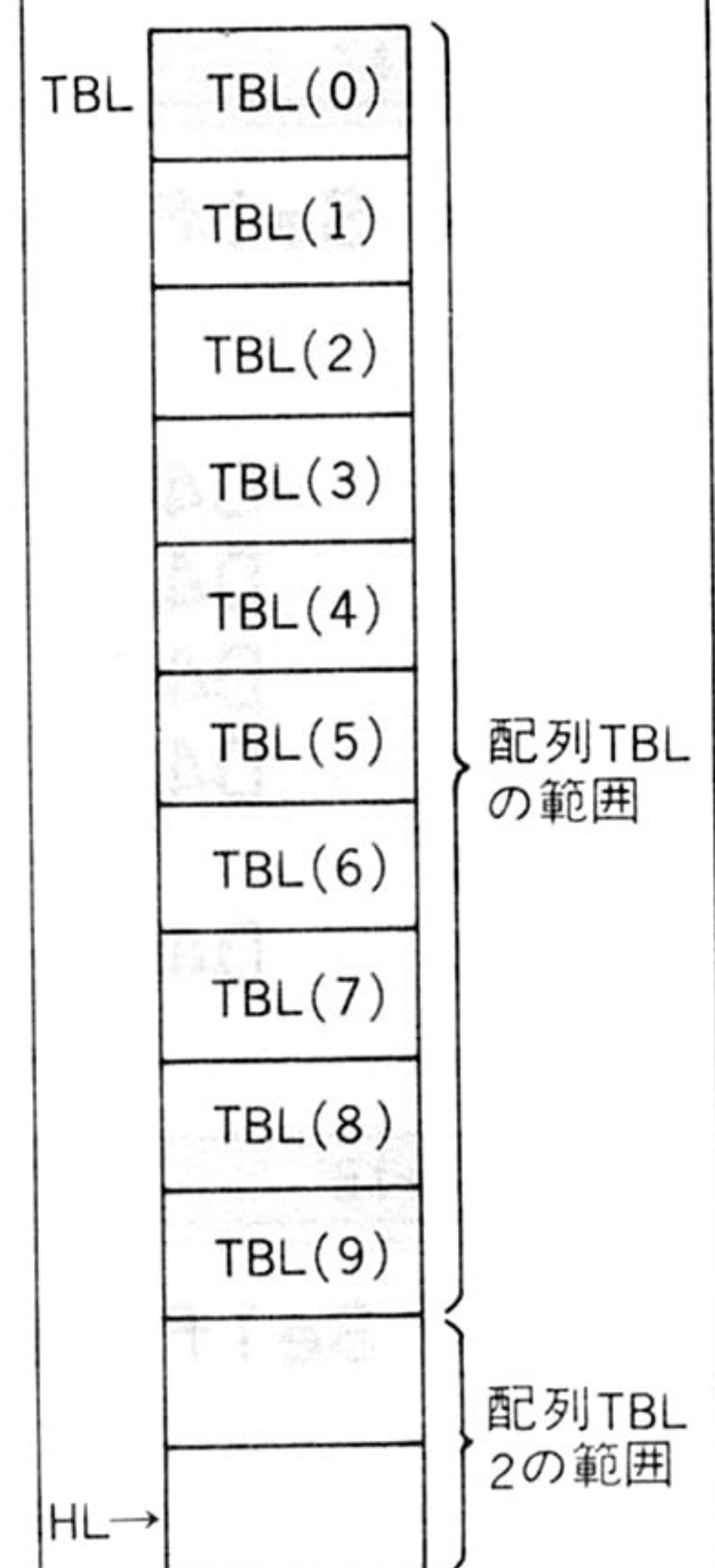
BASICではエラーとなり実行を停止しますが，マシン語では暴走してしまうことも考えられます。

また，同じ働きのプログラムをインデックス・レジスタのＩＸを使ってつくるとP112の**リスト 4-17**，**4-18** のようになります。

また，P112の**リスト 4-19**，P113の **4-20** のようにもできます。

図4-8

TBL	TBL(0)
	TBL(1)
	TBL(2)
	TBL(3)
	TBL(4)
	TBL(5)
	TBL(6)
	TBL(7)
	TBL(8)
	TBL(9)
HL→	

TBL(0)～TBL(9)：配列TBLの範囲
その下の2つ：配列TBL2の範囲

リスト4-15

```
MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                   ORG     0D400H
110:    D400 2109D4            LD      HL,TBL
120:    D403 010500            LD      BC,5
130:    D406 09                ADD     HL,BC
140:    D407 77                LD      (HL),A
150:    D408 C9                RET
160:
170:    D409          TBL:     DEFS    10
```

リスト4-16

```
MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                   ORG     0D400H
110:    D400 2109D4            LD      HL,TBL
120:    D403 010B00            LD      BC,11
```

```
130:    D406 09                    ADD      HL,BC
140:    D407 77                    LD       (HL),A
150:    D408 C9                    RET
160:
170:    D409            TBL:       DEFS     10
180:    D413            TBL2:      DEFS     5
```

リスト4-17

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                       ORG      0D400H
110:    D400 DD2108D4              LD       IX,TBL
120:    D404 DD7705                LD       (IX+5),A
130:    D407 C9                    RET
140:
150:    D408            TBL:       DEFS     10
```

リスト4-18

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                       ORG      0D400H
110:    D400 DD2108D4              LD       IX,TBL
120:    D404 DD7E05                LD       A,(IX+5)
130:    D407 C9                    RET
140:
150:    D408            TBL:       DEFS     10
```

リスト4-19

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                       ORG      0D400H
110:    D400 3A09D4                LD       A,(TBL+5)
120:    D403 C9                    RET
130:
140:    D404            TBL:       DEFS     10
```

リスト4-20

```
 MSX Self Assembler   Rev 1.0     PAGE      1


100:     D400                      ORG       0D400H
110:     D400 3209D4               LD        (TBL+5),A
120:     D403 C9                   RET
130:
140:     D404            TBL:      DEFS      10
```

次に応用として，次のような場合を考えましょう。

配列TBLの中で最大の要素を求める

Bレジスタに要素の数，HLレジスタに配列の先頭アドレス，Cレジスタに今までの最大値を入れます。Cレジスタは初期値を0にします。これで**リスト4-21**のプログラムができます。結果はCレジスタに入ります。

リスト4-21

```
 MSX Self Assembler   Rev 1.0     PAGE      1


100:     D400                      ORG       0D400H
110:     D400 210FD4               LD        HL,TBL
120:     D403 01000A               LD        BC,0A00H
130:     D406 7E         LOOP:     LD        A,(HL)
140:     D407 23                   INC       HL
150:     D408 B9                   CP        C
160:     D409 3801                 JR        C,SKIP
170:     D40B 4F                   LD        C,A
180:     D40C 10F8       SKIP:     DJNZ      LOOP
190:     D40E C9                   RET
200:
210:     D40F            TBL:      DEFS      10
```

今度は要素の大きさが2バイトのときを考えましょう。

2バイトの要素10個の配列を使うことを宣言する

これは1バイトの倍の大きさにすればよいので，P114の**リスト4-22**のように宣言します。配列の要素はP114**図4-9**のように，下位8ビット，上8ビットの順に入れます。

TBL2(5)の値をDEレジスタに入れるには**リスト 4-23**，逆にDEレジスタの値をTBL 2(5)に入れるには**リスト 4-24** のようにします。

これらが**リスト 4-14**， **4-15** と違うところは，要素の番号を入れたBCレジスタを配列の先頭アドレスを入れたHLレジスタに2回足している点です。

図4-9

TBL2(0)の下位
〃　の上位
TBL2(1)の下位
〃　の上位
⋮
TBL2(9)の下位
〃　の上位

リスト4-22

```
 MSX Self Assembler   Rev 1.0     PAGE     1


100:    D400                      ORG      0D400H
110:    D400           TBL2:      DEFS     20
```

リスト4-23

```
 MSX Self Assembler   Rev 1.0     PAGE     1


100:    D400                      ORG      0D400H
110:    D400 210CD4               LD       HL,TBL2
120:    D403 010500               LD       BC,5
130:    D406 09                   ADD      HL,BC
140:    D407 09                   ADD      HL,BC
150:    D408 5E                   LD       E,(HL)
160:    D409 23                   INC      HL
170:    D40A 56                   LD       D,(HL)
180:    D40B C9                   RET
190:
200:    D40C           TBL2:      DEFS     20
```

リスト4-24

```
 MSX Self Assembler   Rev 1.0     PAGE     1


100:    D400                      ORG      0D400H
110:    D400 210CD4               LD       HL,TBL2
120:    D403 010500               LD       BC,5
130:    D406 09                   ADD      HL,BC
140:    D407 09                   ADD      HL,BC
150:    D408 73                   LD       (HL),E
```

```
160:    D409 23                 INC     HL
170:    D40A 72                 LD      (HL),D
180:    D40B C9                 RET
190:
200:    D40C          TBL2:     DEFS    20
```

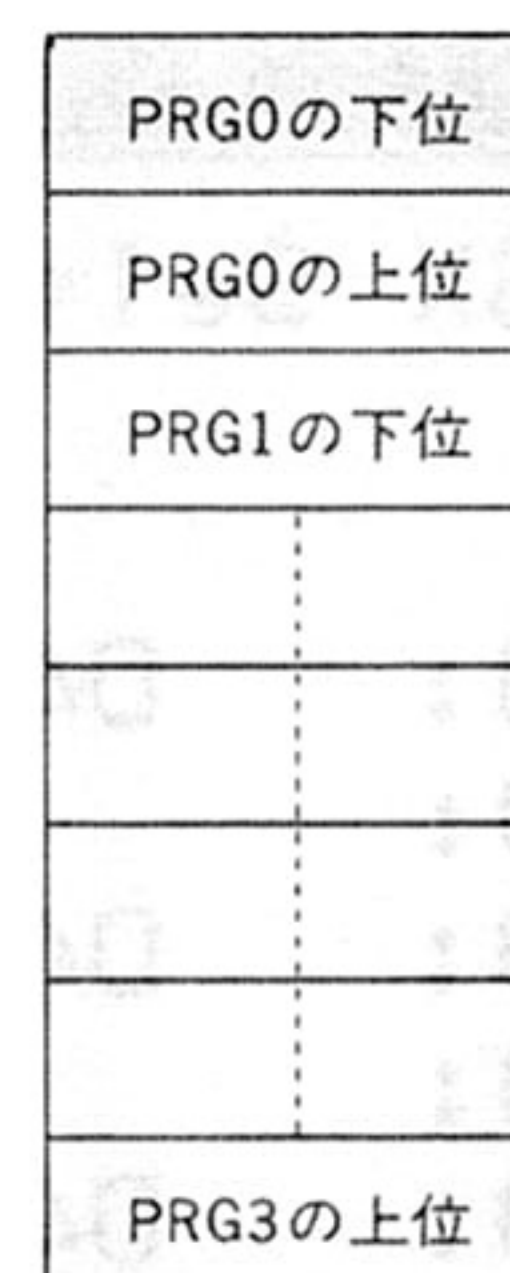
2バイト要素の配列の応用として，次のような場合を考えましょう。

“ON 〈変数〉 GOTO” 文をマシン語で実現する

これは，変数が0ならプログラム0を，1ならプログラム1を…というように，変数の値によって飛び先を変えるプログラムのことで，配列を使ってつくります。

まず **リスト 4-25** のように配列に飛び先アドレスを設定します。ここでDEFWというのは，**リスト 4-13**のDEFSと同じく実行するための命令ではありません。ラベルのアドレスを入れた配列を用意するという意味です。このように飛び先アドレスをいれた配列のことを，ジャンプ・テーブルと呼びます(**図 4-10**)。

図4-10

PRG0の下位
PRG0の上位
PRG1の下位
⋮
⋮
⋮
⋮
PRG3の上位

リスト4-25

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                    ORG     0D400H
110:                  ;
120:    D400 08D4     JPTBL:    DEFW    PRG0
130:    D402 0AD4               DEFW    PRG1
140:    D404 0DD4               DEFW    PRG2
150:    D406 10D4               DEFW    PRG3
160:                  ;
170:    D408 AF       PRG0:     XOR     A
180:    D409 C9                 RET
190:    D40A 3E01     PRG1:     LD      A,1
200:    D40C C9                 RET
210:    D40D 3E02     PRG2:     LD      A,2
220:    D40F C9                 RET
230:    D410 3E03     PRG3:     LD      A,3
240:    D412 C9                 RET
```

この配列から飛び先アドレスを取り，HLレジスタに入れてJP(HL)命令を使うプログラムが **リスト4-26** です。このプログラムを実行するには **リスト4-25** のプログラムが必要です。実行はD450H番地からです。BCレジスタに要素の番号が入っています。これを配列の先頭アドレスの入っているHLレジスタに2回足します。次に，HLレジスタの示すアドレスとその次のアドレスの内容をHLレジスタに入れたいのですが，これは1命令ではできません。

まず，CレジスタにHLレジスタの示すアドレスの内容を入れ，次にHLレジスタを1だけ増します。そして，LD H,(HL)命令とLD L,C命令を使います。これを **リスト4-27** のようにしてはいけません。LD L,(HL)命令を実行すると，HLは違ったアドレスを示すことになってしまうからです。

リスト4-26

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D450                     ORG      0D450H
110:                ;
120:    D400 =      JPTBL        EQU      0D400H
130:                ;
140:    D450 2100D4              LD       HL,JPTBL
150:    D453 010200              LD       BC,2
160:    D456 09                  ADD      HL,BC
170:    D457 09                  ADD      HL,BC
180:    D458 4E                  LD       C,(HL)
190:    D459 23                  INC      HL
200:    D45A 66                  LD       H,(HL)
210:    D45B 69                  LD       L,C
220:    D45C E9                  JP       (HL)
```

リスト4-27

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400 =      JPTBL        EQU      0D400H
110:
120:    D450                     ORG      0D450H
130:    D450 2100D4              LD       HL,JPTBL
140:    D453 010200              LD       BC,2
150:    D456 09                  ADD      HL,BC
160:    D457 09                  ADD      HL,BC
```

```
170:    D458 6E                         LD       L,(HL)
180:    D459 23                         INC      HL
190:    D45A 66                         LD       H,(HL)
200:    D45B E9                         JP       (HL)
```

2.2次元配列

今までは1次元配列を考えてきました。次に2次元配列をマシン語で実現してみましょう。

要素の大きさが1バイトの10×10の2次元配列をマシン語で実現する

まず，メモリ上にどのようにして配列の要素を格納するかを宣言します。メモリを**図4-11**のように使うなら**リスト4-28**前半のように宣言します。これとは別にTBL0，TBL1，…，TBL9のアドレスを格納した配列を**リスト4-28**後半のようにして用意します。

TBL(5，3)の要素をAレジスタに入れるには，HLレジスタにTBL5のアドレスを入れ，あとは1次元配列と同じようにすればよいので，**リスト4-29**のようなプログラムができます。

図4-11

TBL(0,0)
TBL(0,1)
TBL(0,2)
TBL(0,3)
TBL(0,4)
⋮
TBL(0,9)
TBL(1,0)
⋮
⋮
TBL(9,9)

ADRTBLは2バイトの一次元配列なので，DEをHLに2回足します。そして，HLレジスタの示すアドレスの内容をHLレジスタに入れ，BCレジスタをHLレジスタに足します。これでHLレジスタがTBL(5，3)のアドレスを示すようになります。

リスト4-28

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D300                    ORG      0D300H
110:    D300          TBL0:     DEFS     10
120:    D30A          TBL1:     DEFS     10
130:    D314          TBL2:     DEFS     10
140:    D31E          TBL3:     DEFS     10
150:    D328          TBL4:     DEFS     10
160:    D332          TBL5:     DEFS     10
170:    D33C          TBL6:     DEFS     10
180:    D346          TBL7:     DEFS     10
190:    D350          TBL8:     DEFS     10
```

```
200:   D35A            TBL9:     DEFS    10
210:
220:   D364 00D3       ADRTBL:   DEFW    TBL0
230:   D366 0AD3                 DEFW    TBL1
240:   D368 14D3                 DEFW    TBL2
250:   D36A 1ED3                 DEFW    TBL3
260:   D36C 28D3                 DEFW    TBL4
270:   D36E 32D3                 DEFW    TBL5
280:   D370 3CD3                 DEFW    TBL6
290:   D372 46D3                 DEFW    TBL7
300:   D374 50D3                 DEFW    TBL8
310:   D376 5AD3                 DEFW    TBL9
```

リスト4-29

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:   D364 =          ADRTBL    EQU     0D364H
110:
120:   D400                      ORG     0D400H
130:   D400 110500               LD      DE,5
140:   D403 010300               LD      BC,3
150:   D406 2164D3               LD      HL,ADRTBL
160:   D409 19                   ADD     HL,DE
170:   D40A 19                   ADD     HL,DE
180:   D40B 5E                   LD      E,(HL)
190:   D40C 23                   INC     HL
200:   D40D 66                   LD      H,(HL)
210:   D40E 6B                   LD      L,E
220:   D40F 09                   ADD     HL,BC
230:   D410 7E                   LD      A,(HL)
240:   D411 C9                   RET
```

ものにしよう実践テクニック

3 文字列サーチ

次は，文字列をサーチするプログラムをつくりましょう。ここでは，HLレジスタが示すアドレスからの文字列が“LIST”なら0，“RUN”なら1，“LOAD”なら2，“SAVE”なら3，その他ならFFHをAレジスタに入れて戻るルーチンとします。

これは，アセンブラなどを自作しようとするときに必要になります。

テーブルに，

文字列，0，値

という形で，LIST，RUNなどの値を並べて書きます。テーブルの終わりは，

0，0FFH

とします。これで，テーブルにない文字列のときはAレジスタにFFHが入ります。

プログラムは **リスト 4-30** のようになります。

DEレジスタでテーブルを示し，その値が0なら文字列が一致したとみなします。0でないときは，HLの示す番地の内容と比較し，これが等しくなくなるまで繰り返します。等しくなければ，テーブルの次の文字列“LIST”の次なら“RUN”と比較します。このとき，HLレジスタの値をもとに戻さなくてはなりません。

テーブルの最後は0，FFHですから，HLの示す文字列とそれ以上比較することなく，値をFFHとしてAレジスタに入れます。

リスト4-30

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    D400                    ORG      0D400H
110:    D400 1118D4             LD       DE,TBL
120:    D403 E5       LOOP:     PUSH     HL
130:    D404 1A       LOOP1:    LD       A,(DE)
140:    D405 13                 INC      DE
150:    D406 B7                 OR       A
160:    D407 280D               JR       Z,FIND
170:    D409 BE                 CP       (HL)
180:    D40A 23                 INC      HL
190:    D40B 28F7               JR       Z,LOOP1
```

```
200:    D40D E1                      POP     HL
210:    D40E 1A         NEXT:        LD      A,(DE)
220:    D40F 13                      INC     DE
230:    D410 B7                      OR      A
240:    D411 20FB                    JR      NZ,NEXT
250:    D413 13                      INC     DE
260:    D414 18ED                    JR      LOOP
270:    D416 1A         FIND:        LD      A,(DE)
280:    D417 C9                      RET
290:
300:    D418 4C495354   TBL:         DEFM    ´LIST´
310:    D41C 0000                    DEFB    0,0
320:    D41E 52554E                  DEFM    ´RUN´
330:    D421 0001                    DEFB    0,1
340:    D423 4C4F4144                DEFM    ´LOAD´
350:    D427 0002                    DEFB    0,2
360:    D429 53415645                DEFM    ´SAVE´
370:    D42D 0003                    DEFB    0,3
380:    D42F 00FF                    DEFB    0,0FFH
```

ものにしよう実践テクニック

4 乗除算のプログラム

1. 簡単なかけ算，割り算

Z80にはかけ算，割り算の命令がありませんから，そのためのサブルーチンをつくらなくてはなりません。

リスト 4-31 は足し算の繰り返しでかけ算をしています。

DEレジスタに被乗数，BCレジスタに乗数を入れてこのサブルーチンをコールすると，BC回だけDEレジスタをHLレジスタに足します。ループの先頭で終了の判定をしていますから，BCレジスタが0のときは一度もADD HL,DE命令を実行しません。

このサブルーチンを使うときは，BC，DE，HLの各レジスタは0～65535までの数を表わすものとします。

かけ算の答えが65535を超えると正しい結果を得ることができません。

割り算のサブルーチンは被除数をHLレジスタに，除数をDEレジスタに入れ，HLレジスタからDEレジスタを何回引くことができるかを数えることによりつくります。これは **リスト 4-32** のようになります，割り算の答はBCレジスタに入ります。

はじめに，BCレジスタに－1を入れます。これは，ループの先頭

でINC BC命令をするするためです。DEレジスタが0のときは答が65535となります。

次に，Cフラグが立つまでHLレジスタからDEレジスタを引きます。このループを実行する前にORE命令を実行してCフラグを0にしています。

リスト4-31

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    D400                    ORG      0D400H
110:    D400 210000   MULT:     LD       HL,0
120:    D403 79       LOOP:     LD       A,C
130:    D404 B0                 OR       B
140:    D405 2804               JR       Z,EXIT
150:    D407 19                 ADD      HL,DE
160:    D408 0B                 DEC      BC
170:    D409 18F8               JR       LOOP
180:    D40B C9       EXIT:     RET
```

リスト4-32

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    D400                    ORG      0D400H
110:    D400 01FFFF             LD       BC,-1
120:    D403 7A                 LD       A,D
130:    D404 B3                 OR       E
140:    D405 2805               JR       Z,EXIT
150:    D407 03       LOOP:     INC      BC
160:    D408 ED52               SBC      HL,DE
170:    D40A 30FB               JR       NC,LOOP
180:    D40C C9       EXIT:     RET
```

2.効率のよいかけ算

かけ算を足し算の繰り返しで行なうと，最大で65535回もループをまわることになります。P122の**図 4-12**のように2進数の筆算を行なうようにすれば，もっと効率のよいかけ算のプログラムを組むことができます。つまり，被乗数の0111と乗数の1桁め，01110と乗数の2桁め，…というようにそれぞれをかけた結果の和がかけ算の答となります。

これをプログラムにするとP122の**リスト 4-33**のようになります。Bレジスタに被乗数，Cレジスタに乗数を入れます。INC C, DEC C

命令はCレジスタが0かどうか調べています。SRL C命令でCレジスタの最下位ビットをCフラグに送り，Cフラグが立っていればAレジスタにBレジスタを足します。

SLA B命令で被乗数を2倍します。このループをCレジスタが0になるまで繰り返します。このプログラムの答えはAレジスタに入ります。

では，同様にして16ビット×16ビットの計算をするサブルーチンを考えてみましょう。

リスト4-34がそのサブルーチンです。乗数をBCレジスタ，被乗数をDEレジスタに入れて，このサブルーチンをコールすると答がHLレジスタに入ります。

16ビットのINC,DEC命令ではフラグが変化しないので，BCレジスタの内容が0かどうかを調べるのにAレジスタを使用しています。BCレジスタの最下位ビットをSRL B,RR C命令でCフラグに送り（図

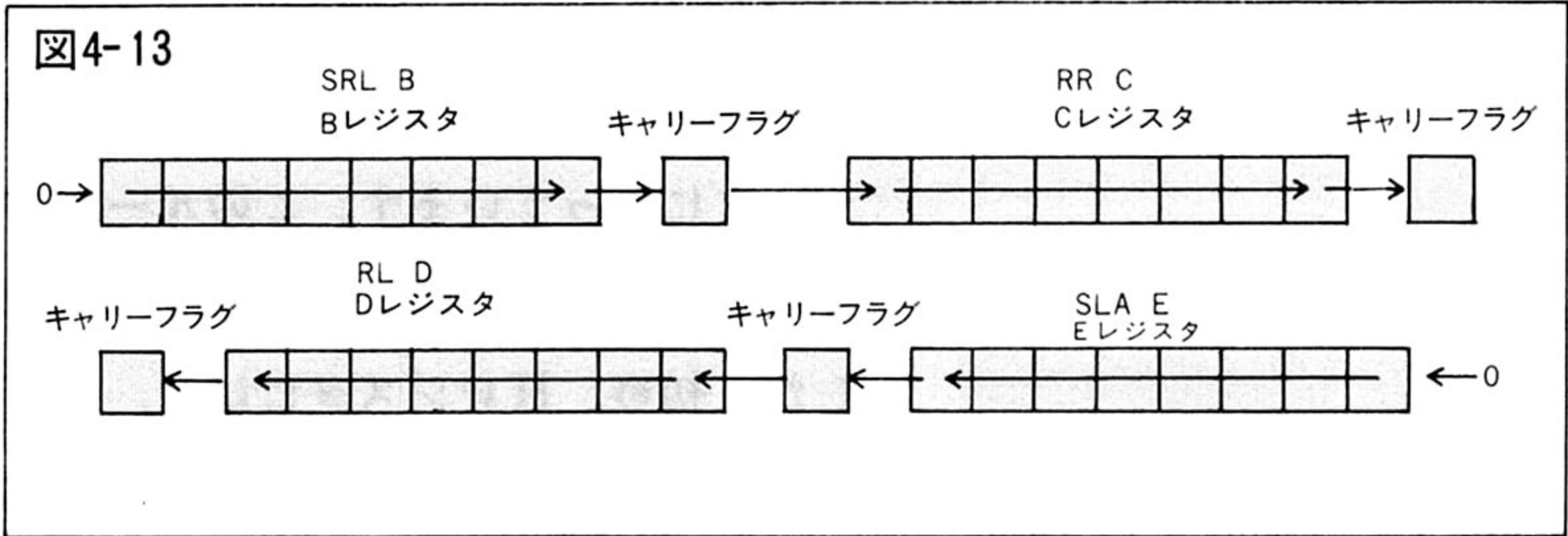
図4-12

```
         0 1 1 1
       × 1 0 1 0  ──────────────┐
      ──────────                ↓
         0 0 0 0  ←──     0 1 1 1× 0
  +    0 1 1 1    ←──   0 1 1 1 0× 1
  +  0 0 0 0      ←── 0 1 1 1 0 0× 0
  + 0 1 1 1       ←──0 1 1 1 0 0 0× 1
  ──────────
   1 0 0 0 1 1 0
```

リスト4-33

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:   D400                      ORG     0D400H
110:   D400 010A07               LD      BC,070AH
120:   D403 AF                   XOR     A
130:   D404 0C        LOOP:      INC     C
140:   D405 0D                   DEC     C
150:   D406 2809                 JR      Z,EXIT
160:   D408 CB39                 SRL     C
170:   D40A 3001                 JR      NC,SKIP
180:   D40C 80                   ADD     A,B
190:   D40D CB20      SKIP:      SLA     B
200:   D40F 18F3                 JR      LOOP
210:   D411 C9        EXIT:      RET
```

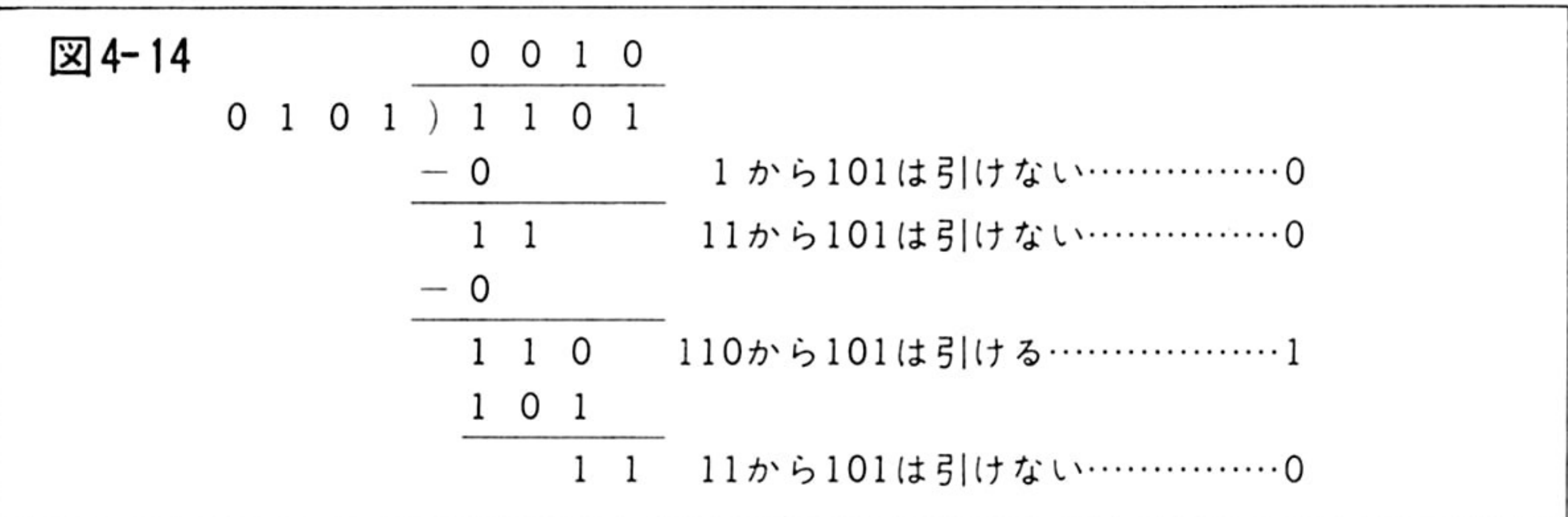
4-13)，Cフラグが立てばHLレジスタにDEレジスタを足します。DEレジスタを2倍するには,SLA E，RL D命令を使います(図4-13)。このループをBCレジスタが0になるまで続けます。

図4-13

リスト4-34

```
MSX Self Assembler   Rev 1.0     PAGE     1


100:    D400                     ORG      0D400H
110:    D400 210000   MULT:      LD       HL,0
120:    D403 79       LOOP:      LD       A,C
130:    D404 B0                  OR       B
140:    D405 280D                JR       Z,EXIT
150:    D407 CB38                SRL      B
160:    D409 CB19                RR       C
170:    D40B 3001                JR       NC,SKIP
180:    D40D 19                  ADD      HL,DE
190:    D40E CB23     SKIP:      SLA      E
200:    D410 CB12                RL       D
210:    D412 18EF                JR       LOOP
220:    D414 C9       EXIT:      RET
```

3.効率のよい割り算

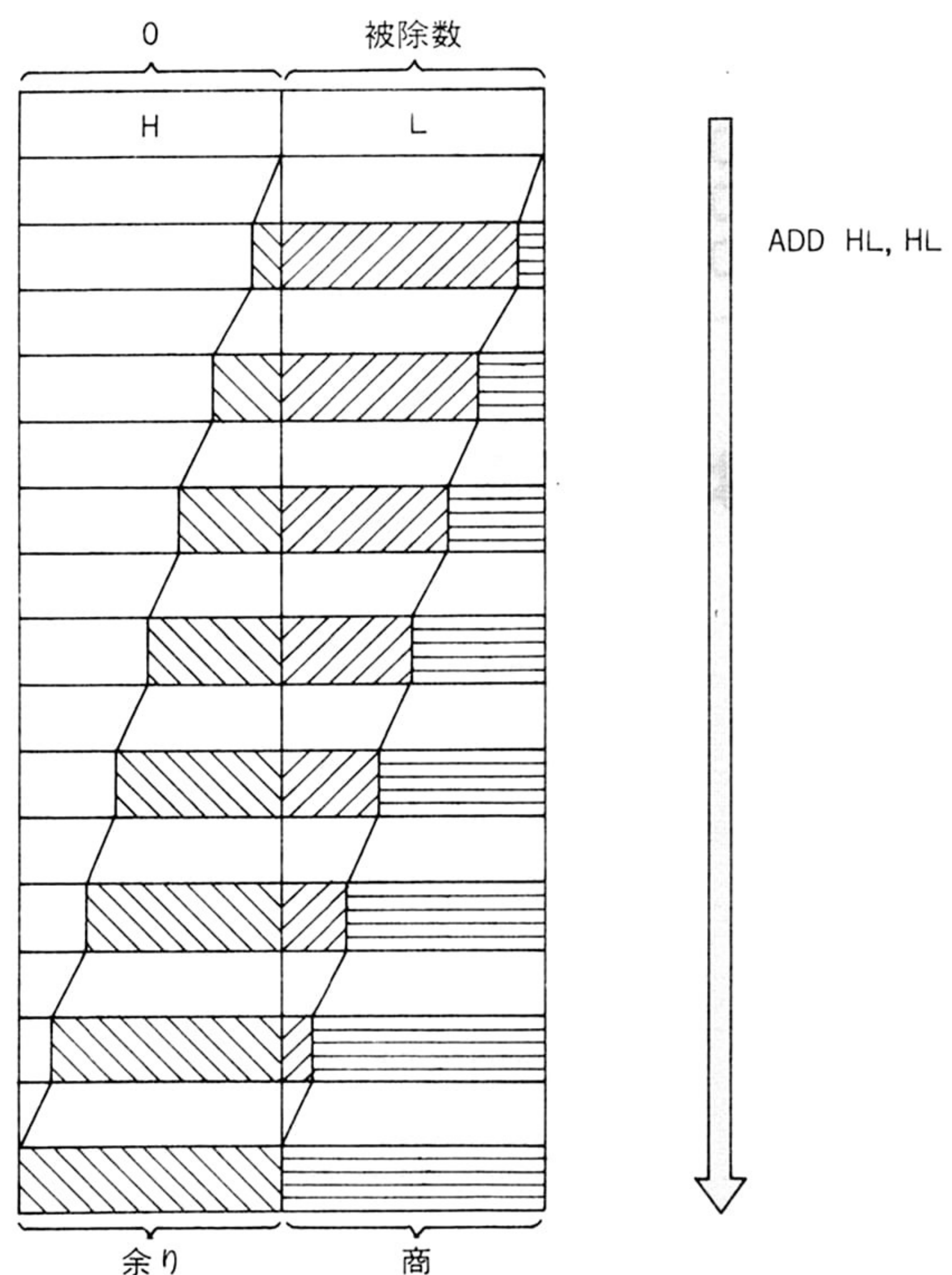
割り算も筆算の方法を使うと効率のよいプログラムがつくれます。図4-14は，2進数の割り算を筆算で計算したものです。被除数の最上位の桁から除数を引くことができれば，答の最上位を1に，引くことができなければ答の最上位を0にします。引いた残りを余りとします。これを8回繰り返すと8bit÷8bitの割り算ができます。

図4-14

被除数をLレジスタ，除数をCレジスタ，余りをHLジスタとして，100÷16のプログラムをつくると**リスト4-35**のようになります。LD HL, 100でHレジスタに0，Lレジスタに100を入れます。LD BC, 810HでCレジスタに16を，Bレジスタに繰り返し回数の8を入れます。次のADD HL, HL命令で，Lレジスタの被除数の最上位ビットをHレジスタに送っています。このループ中のHLレジスタの使いかたはちょっと複雑です。**図4-15**にHLレジスタの使いかたを図示してみました。初め，Hレジスタには0，Lレジスタには被除数を入れておきます。ADD HL, HL命令を実行するたびに，Hレジスタ，Lレジスタのそれぞれの下位ビットから順に，余りと商が入ってきます。8回ループを繰り返すとHレジスタは余り，Lレジスタは商を持つことになります。

ADD HL, HL命令でHレジスタに送られたものからCレジスタを引きます。このとき，引くことができればINC LレジスタでLレジスタの最下位ビットを1にします。ADD HL, HL命令を実行するとLレジスタの最下位ビットは必ず0になるからです。

このサブルーチンではCレジスタ，Lレジスタ，Hレジスタ，Bレジスタはそれぞれ0～255を表わしています。

図4-15

0
被除数
H
L
ADD HL, HL
余り
商

リスト4-35

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                     ORG      0D400H
110:    D400 216400              LD       HL,100
120:    D403 011008              LD       BC,810H
130:    D406 29        LOOP:     ADD      HL,HL
140:    D407 7C                  LD       A,H
150:    D408 91                  SUB      C
160:    D409 3802                JR       C,SKIP
170:    D40B 2C                  INC      L
180:    D40C 67                  LD       H,A
190:    D40D 10F7      SKIP:     DJNZ     LOOP
200:    D40F C9                  RET
```

8ビット÷8ビットが理解できれば，次のようなときも同じ方法が使えます。

符号なしの16ビット÷16ビットの計算をする

DEレジスタに被除数，BCレジスタに除数を入れ，コールするとHLレジスタに余り，DEレジスタに商が入るサブルーチンは**リスト4-36**のようになります。

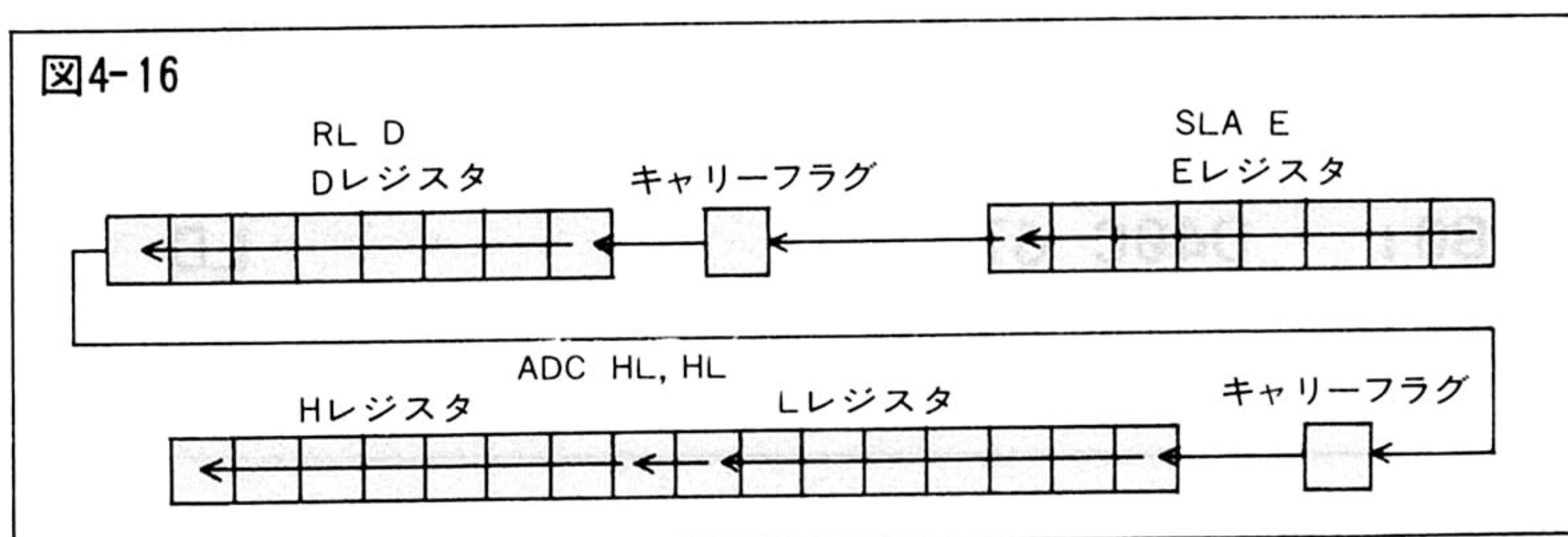
リスト4-36

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                     ORG      0D400H
110:    D400 210000    DIV:      LD       HL,0
120:    D403 D9                  EXX
130:    D404 0610                LD       B,16
140:    D406 D9        LOOP:     EXX
150:    D407 CB23                SLA      E
160:    D409 CB12                RL       D
170:    D40B ED6A                ADC      HL,HL
180:    D40D ED42                SBC      HL,BC
190:    D40F 3803                JR       C,SKIP
200:    D411 1C                  INC      E
210:    D412 1801                JR       SKIP1
220:    D414 09        SKIP:     ADD      HL,BC
230:    D415 D9        SKIP1:    EXX
240:    D416 10EE                DJNZ     LOOP
250:    D418 D9                  EXX
260:    D419 C9                  RET
```

ループを16回繰り返すために裏レジスタのBレジスタを使用します。SLA E, RL D, ADC HL, HL命令でDEレジスタの最上位ビットをHLレジスタに送っています(**図4-16**)。そして，HLレジスタからBCレジスタをSBC HL, BC命令で引き算します。

HLレジスタは初めに0にしてあるので，ADC HL, HL命令を実行後，常にCフラグが0になります。SBC HL, BCの前にCフラグをクリアするためのOR A命令は必要ありません。

図4-16

RL D
Dレジスタ
キャリーフラグ
SLA E
Eレジスタ
ADC HL, HL
Hレジスタ
Lレジスタ
キャリーフラグ

引き算の後，Cフラグが立たなければEレジスタの最下位ビットを1にします。Cフラグが立ったときは，ADD HL, BC命令でHLレジスタをもとに戻します。

以上のループを16回繰り返します。このサブルーチンでは，HL, DE, BCはそれぞれ0〜65535の値を表わしています。

このサブルーチンを利用して符号つきの割り算のサブルーチンをつくってみましょう。

符号つき16ビット÷16ビットの計算をするサブルーチンをつくる

被除数と除数の符号から，商の符号がわかります。そして，商の絶対値は被除数の絶対値÷除数の絶対値で求めることができます。

そこで，DEレジスタとBCレジスタの符号が同じならCフラグを立て違うならCフラグをオフにするサブルーチンをつくります。これは**リスト4-37**のようになります。またHLレジスタの絶対値をHLレジスタに入れるプログラムは**リスト4-38**のようになります。

これらを使って符号つきの割り算を計算するプログラムは**リスト4-39**のようになります。このプログラムの実行には**リスト4-36，4-37，4-38**のプログラムが必要です。実行はD450H番地からです。

DEレジスタの絶対値÷BCレジスタの絶対値の商をDEレジスタに入れ，商が負のときはDEレジスタも負にします。なお，このプログラムでは余りは正しく求まりません。

リスト4-37

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D430                    ORG     0D430H
110:    D430 78       CPSBD:    LD      A,B
120:    D431 B7                 OR      A
130:    D432 FA3CD4             JP      M,MI1
140:    D435 7A                 LD      A,D
150:    D436 B7                 OR      A
160:    D437 F241D4             JP      P,PLUS
170:    D43A 1806               JR      MINUS
180:    D43C 7A       MI1:      LD      A,D
190:    D43D B7                 OR      A
200:    D43E F242D4             JP      P,MINUS
210:    D441 37       PLUS:     SCF
220:    D442 C9       MINUS:    RET
```

リスト4-38

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D420                    ORG     0D420H
110:    D420 7C       ABS:      LD      A,H
120:    D421 B7                 OR      A
130:    D422 F22BD4             JP      P,EXIT
140:    D425 2F                 CPL
150:    D426 67                 LD      H,A
160:    D427 7D                 LD      A,L
170:    D428 2F                 CPL
180:    D429 6F                 LD      L,A
190:    D42A 23                 INC     HL
200:    D42B C9       EXIT:     RET
```

リスト4-39

```
 MSX Self Assembler  Rev 1.0    PAGE    1

100:    D400 =        DIV       EQU     0D400H
110:    D420 =        ABS       EQU     0D420H
120:    D430 =        CPSBD     EQU     0D430H
130:
140:    D450                    ORG     0D450H
150:    D450 CD30D4   DIVS:     CALL    CPSBD
160:    D453 F5                 PUSH    AF
170:    D454 69                 LD      L,C
180:    D455 60                 LD      H,B
190:    D456 CD20D4             CALL    ABS
```

```
200:    D459 4D                 LD      C,L
210:    D45A 44                 LD      B,H
220:    D45B EB                 EX      DE,HL
230:    D45C CD20D4             CALL    ABS
240:    D45F EB                 EX      DE,HL
250:    D460 CD00D4             CALL    DIV
260:    D463 F1                 POP     AF
270:    D464 3807               JR      C,EXITD
280:    D466 7A                 LD      A,D
290:    D467 2F                 CPL
300:    D468 57                 LD      D,A
310:    D469 7B                 LD      A,E
320:    D46A 2F                 CPL
330:    D46B 5F                 LD      E,A
340:    D46C 13                 INC     DE
350:    D46D C9         EXITD:  RET
```

4. 定数のかけ算と割り算

たとえば，DEレジスタを3倍したいときは，**リスト4-40**のようにします。このプログラムを実行するには**リスト4-34**のプログラムが必要です。実行はD420H番地からです。しかし，**リスト4-41**のようにしたほうが速くなります。HLレジスタを10倍したいときは**リスト4-42**のようにします。では15倍したいときはどうしますか？ **リスト4-43**を見てください。HLレジスタの内容をDEレジスタにコピーし，HLを16倍します。そしてHLレジスタからDEレジスタを引けば15倍したことになります。

3で割る，5で割るなどの場合はかけ算のときのようなうまい方法はありません。

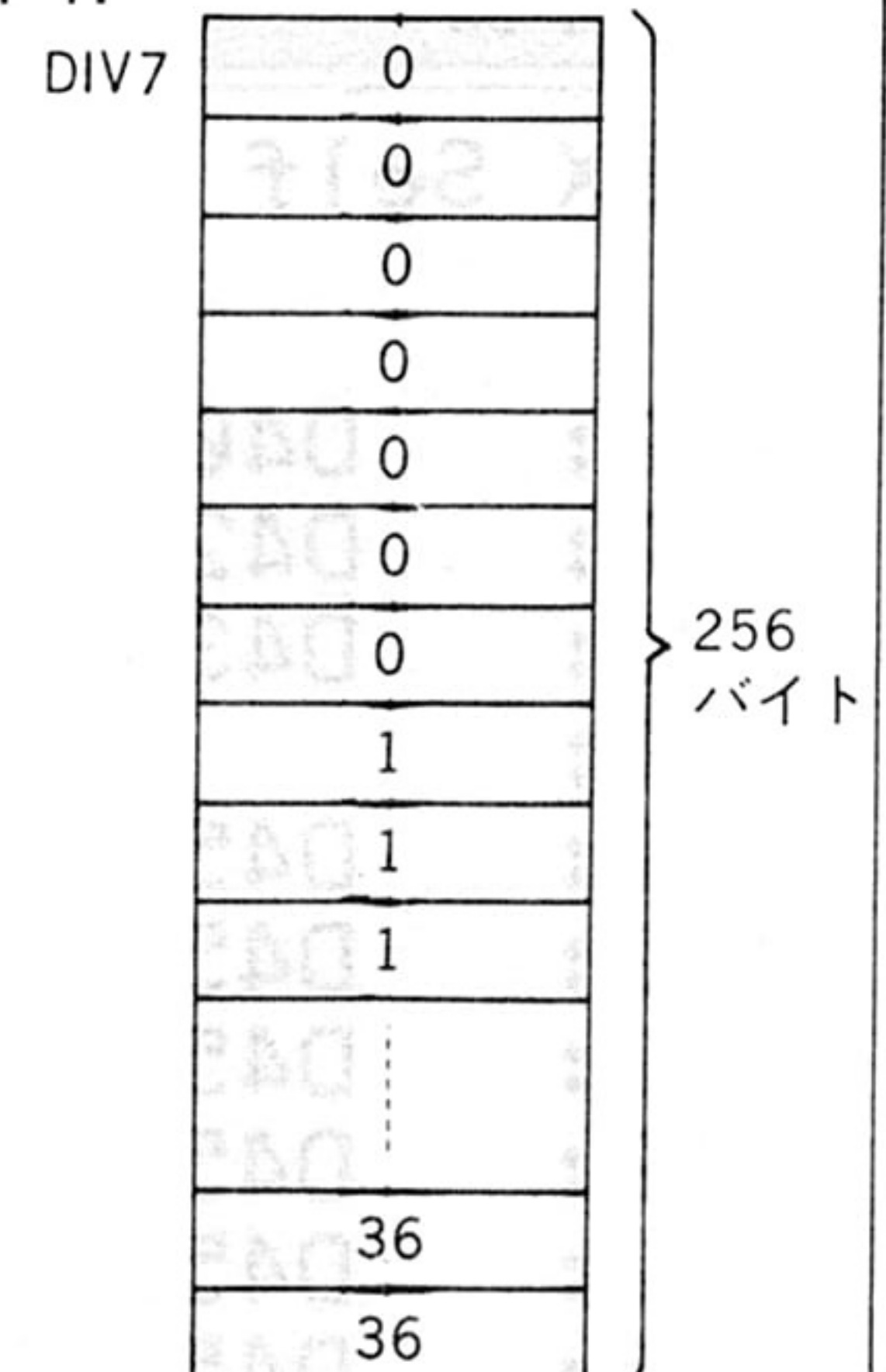
2,4,8など2^nで表わせる数が除数のときは，n回右へシフトすればよいので簡単です。また2^nで割った余りを求めるには2^n-1とAND演算をします。Lレジスタを8で割った余りを求めるプログラムはP130の**リスト4-44**のようになります。

スピードが問題になるときは，配列を使う方法もあります。

図4-17のように，あらかじめ7で割った商を配列D1V7に入れておけば7で割る割り算はP130の**リスト4-45**のようになります。

図4-17

DIV7	
	0
	0
	0
	0
	0
	0
	0
	1
	1
	1
	⋮
	36
	36

（256バイト）

このサブルーチンを使うときはBレジスタに０，Ｃレジスタに被除数を入れコールします。商はＡレジスタに求まります。

リスト4-40

```
 MSX Self Assembler   Rev 1.0      PAGE      1


100:    D400 =          MULT        EQU       0D400H
110:
120:    D420                        ORG       0D420H
130:    D420 010300                 LD        BC,3
140:    D423 CD00D4                 CALL      MULT
150:    D426 C9                     RET
```

リスト4-41

```
 MSX Self Assembler   Rev 1.0      PAGE      1


100:    D400                        ORG       0D400H
110:    D400 54                     LD        D,H
120:    D401 5D                     LD        E,L
130:    D402 29                     ADD       HL,HL
140:    D403 19                     ADD       HL,DE
150:    D404 C9                     RET
```

リスト4-42

```
 MSX Self Assembler   Rev 1.0      PAGE      1


100:    D400                        ORG       0D400H
110:    D400 29                     ADD       HL,HL
120:    D401 54                     LD        D,H
130:    D402 5D                     LD        E,L
140:    D403 29                     ADD       HL,HL
150:    D404 29                     ADD       HL,HL
160:    D405 19                     ADD       HL,DE
170:    D406 C9                     RET
```

リスト4-43

```
 MSX Self Assembler   Rev 1.0      PAGE      1


100:    D400                        ORG       0D400H
110:    D400 5D                     LD        E,L
120:    D401 54                     LD        D,H
130:    D402 29                     ADD       HL,HL
140:    D403 29                     ADD       HL,HL
```

```
150:    D404 29                     ADD     HL,HL
160:    D405 29                     ADD     HL,HL
170:    D406 B7                     OR      A
180:    D407 ED52                   SBC     HL,DE
190:    D409 C9                     RET
```

リスト4-44

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D400                        ORG     0D400H
110:    D400 3E07                   LD      A,7
120:    D402 A5                     AND     L
130:    D403 C9                     RET
```

リスト4-45

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    D300 =          DIV7        EQU     0D300H
110:
120:    D400                        ORG     0D400H
130:    D400 2100D3                 LD      HL,DIV7
140:    D403 09                     ADD     HL,BC
150:    D404 7E                     LD      A,(HL)
160:    D405 C9                     RET
```

5.10進数表示

これまでつくってきた割り算ルーチンを応用して次のプログラムを考えてみましょう。

HLレジスタを10進数で表示する

このためには，HLを10000で割って商を表示し，その余りを1000で割って商を表示…というように5回繰り返します。

プログラムにすると**リスト4-46**のようになります。このプログラムを実行するには**リスト4-36**のプログラムが必要です。実行はD430H番地からです。

まず，BCレジスタに10000を入れます。次にHLレジスタをBCレジスタで割り，その商をASCIIコードに変換し，BIOSの1文字表示ルーチンで表示します。BCレジスタを10で割り，商が0なら終了します。そうでないときはループを繰り返します。

このサブルーチンはHLを符号なしの数として扱いますから，0～65535までの値を表示します。符号つきの数として－32768～32767までを表示するサブルーチンはP132の**リスト4-47**のようになります。つまり，HLが負のときは－(マイナス)を表示した後，HLの絶対値を表示します。このプログラムを実行するには，**リスト4-36**，**4-38**，**4-46**のプログラムが必要です。実行はD450H番地からです。

今度は逆に，次のようなときを考えてみましょう。

メモリ中の10進数の文字列を2進数にして，HLレジスタに入れる

文字列の終わりは0で表わすことにします。まず文字列の先頭の1文字を2進数にし,次の文字が0でないときはHLレジスタを10倍して，その文字を2進数にして足します。

たとえば“30172”をHLに入れるプログラムは**リスト4-48**のようになります。

この章ではいくつかの短いプログラムをつくってみました。自分でプログラムをつくるときに，これを参考にしたり改造したりして役立ててください。

リスト4-46

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    00A2 =         CHPUT       EQU      00A2H
110:    D400 =         DIV         EQU      0D400H
120:
130:    D430                       ORG      0D430H
140:    D430 011027    DECOUT:     LD       BC,10000
150:    D433 EB        LOOP:       EX       DE,HL
160:    D434 CD00D4                CALL     DIV
170:    D437 7B                    LD       A,E
180:    D438 C630                  ADD      A,30H
190:    D43A CDA200                CALL     CHPUT
200:    D43D 79                    LD       A,C
210:    D43E 3D                    DEC      A
220:    D43F 280E                  JR       Z,EXIT
230:    D441 59                    LD       E,C
240:    D442 50                    LD       D,B
250:    D443 010A00                LD       BC,10
260:    D446 E5                    PUSH     HL
270:    D447 CD00D4                CALL     DIV
280:    D44A E1                    POP      HL
290:    D44B 4B                    LD       C,E
```

```
300:    D44C 42                 LD      B,D
310:    D44D 18E4               JR      LOOP
320:    D44F C9        EXIT:    RET
```

リスト4-47

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    00A2 =         CHPUT    EQU     00A2H
110:    D420 =         ABS      EQU     0D420H
120:    D430 =         DECOUT   EQU     0D430H
130:
140:    D450                    ORG     0D450H
150:    D450 7C                 LD      A,H
160:    D451 E680               AND     80H
170:    D453 2808               JR      Z,SKIP
180:    D455 3E2D               LD      A,'-'
190:    D457 CDA200             CALL    CHPUT
200:    D45A CD20D4             CALL    ABS
210:    D45D CD30D4    SKIP:    CALL    DECOUT
220:    D460 C9                 RET
```

リスト4-48

```
 MSX Self Assembler  Rev 1.0    PAGE     1


100:    D400                    ORG     0D400H
110:    D400 210000             LD      HL,0
120:    D403 111AD4             LD      DE,STR
130:    D406 1A        LOOP:    LD      A,(DE)
140:    D407 13                 INC     DE
150:    D408 B7                 OR      A
160:    D409 280E               JR      Z,EXIT
170:    D40B 29                 ADD     HL,HL
180:    D40C 4D                 LD      C,L
190:    D40D 44                 LD      B,H
200:    D40E 29                 ADD     HL,HL
210:    D40F 29                 ADD     HL,HL
220:    D410 09                 ADD     HL,BC
230:    D411 D630               SUB     30H
240:    D413 4F                 LD      C,A
250:    D414 0600               LD      B,0
260:    D416 09                 ADD     HL,BC
270:    D417 18ED               JR      LOOP
280:    D419 C9        EXIT:    RET
290:
300:    D41A 33303137  STR:     DEFM    '30172'
310:    D41F 00                 DEFB    0
```

5 ものにしよう実践テクニック MSX-BASICのBCD

BCDとは，4ビットごとに10進数の0～9を表わす方法です(図4-18)。MSX－BASICでは，数値の計算をBCDで行なっています。**リスト4-49**は，BASICにおいて数値をどのようにメモリに記憶しているか（内部表現）を調べるプログラムです。

このプログラムを実行し0を入力すると，メモリの8バイトはすべて0と表示されます(**図4-19**)。1を入力すると順に41，10,0，…，0となります。

数値，たとえば1257を表わすのに，1.257×10^3というかたちで表わすことがあります。このとき，1.257を仮数，10^3の3を指数と呼びます。

リスト4-49

```
10 INPUT A
20 PRINT:PRINT "A=";A
30 B=VARPTR(A)
40 FOR I=0 TO 7
50 PRINT HEX$(PEEK(B+I));" ";
60 NEXT
70 PRINT
80 GOTO 10
```

図4-18

0000	0
0001	1
0010	2
0011	3
0100	4
0101	5
0110	6
0111	7
1000	8
1001	9
1010	使わない
1011	使わない
1100	使わない
1101	使わない
1110	使わない
1111	使わない

図4-19

```
A= 0
0 0 0 0 0 0 0 0

A= 1
41 10 0 0 0 0 0 0

A= 1234
44 12 34 0 0 0 0 0

A=-1
C1 10 0 0 0 0 0 0

A=-1234
C4 12 34 0 0 0 0 0

A= .5
40 50 0 0 0 0 0 0

A= .1
40 10 0 0 0 0 0 0
```

MSX－BASICの数値の内部表現では，最初の1バイトが指数＋41Hの値を持ち，残りの7バイトで仮数の14桁を表わします。

指数に41Hをたすのは，10^{-1}は40H，10^{-2}は3FHというように負の指数も簡単に扱えるようにするためです。仮数は1桁めと2桁めの間に必ず小数点がくるようになっています。つまり，0.02は2.0×10^{-2}と変換します。また負の数は最初の1バイト（指数部）の最上位ビットを1にすることで表現しています。

たとえば－1234は-1.234×10^{3}と変換し，最初の1バイトは3＋41Hで44H，さらに負であることを表現するため最上位ビットを1にしてC4Hとなり，仮数は1234となります。

リスト4-49のプログラムで，いろいろと試してください。

5章 つなげてしまおう BASICとマシン語

BASICだって裸にすればマシン語で書かれている。BASICをマシン語として見てみると，なかなかおもしろいことが見えてくる。BASICとマシン語のおもしろい関係を，ここで紹介。BASICのマニュアルではわからなかったことがわかるようになる。

つなげてしまおうBASICとマシン語

1 マシン語としてみたBASIC

マシン語で書かれたプログラムは，それを単体で動作させることもBASICのプログラムと組み合わせて使うこともあります。特にMSXのような「BASICマシン」でマシン語プログラムを利用しようとするときには，後者の方法によらざるを得ません。

メモリの中をのぞいてみると，BASICのテキスト（BASICプログラムの命令）もマシン語と同じように格納されているのがわかります。もしユーザーのマシン語プログラムが誤動作してこのテキストを書きかえてしまうと，BASICインタープリタには正しいBASICテキストとして認識されなくなるでしょう。逆にテキストがRAMに「書きかえ可能」な形で格納されていることを積極的に利用することも価値があるかもしれません。

また，MSX-BASICの命令にも，マシン語（あるいはメモリ内容）を操作する目的のものがいくつかあります。これらの命令を習得することはBASICとマシン語とのリンク（結合）をさせるときに役立つでしょう。この項では，マシン語的にみたBASICの紹介をしてみようと思います。

1. メモリ内のBASICプログラム

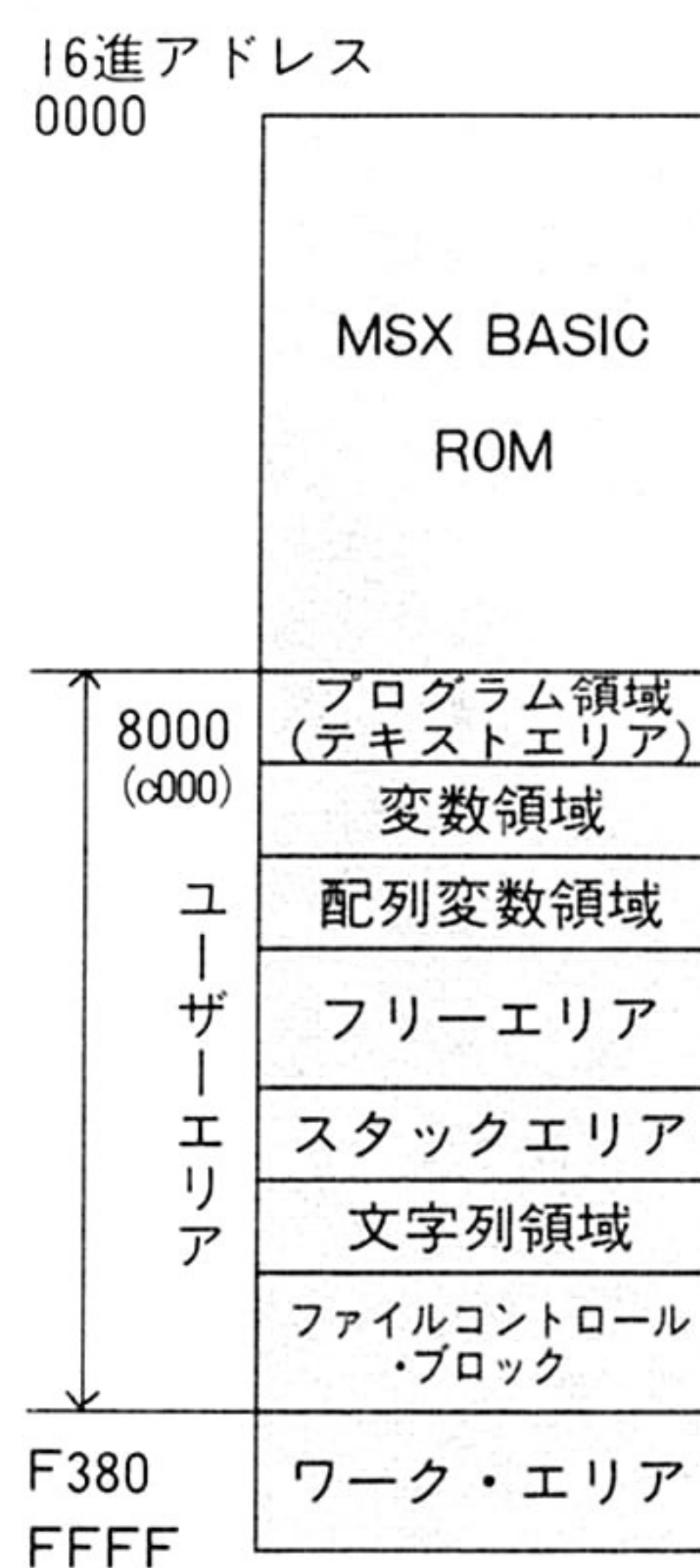
①メモリ・マップ

MSX-BASICを使っているときのメモリの状態はどんなものでしょう。MSXに付属してきた「MSXユーザーズ・マニュアル」の最後にあるメモリ・マップを参照してください（図5-1）。

Z80の64Kバイトのメモリ空間のうち，半分の32Kバイト（0000H～7FFFH）はMSX-BASICが収められているROMにあてられています。ROMですから当然，書き込みは不可能ですね。

図5-1　メモリ・マップ

16進アドレス		内容
0000		MSX BASIC ROM
8000 (c000)	ユーザーエリア	プログラム領域（テキストエリア）
	ユーザーエリア	変数領域
	ユーザーエリア	配列変数領域
	ユーザーエリア	フリーエリア
	ユーザーエリア	スタックエリア
	ユーザーエリア	文字列領域
	ユーザーエリア	ファイルコントロール・ブロック
F380		ワーク・エリア
FFFF		

残った32KバイトがRAMになるエリアです。あなたのMSXが32KRAMであれば，このエリアすべて（8000H～FFFFH）にRAMが装備されています。もし16KRAMのマシンで

あれば，RAMはC000H～FFFFHまでで，8000H～BFFFHまでには何もありません。しかも，RAMエリアの最上位部分にあたるF380H～FFFFHまではBASICインタープリタが使うワーク・エリアですから，ユーザーがプログラムなどのために使うことはできません。つまり，ユーザーのためのエリア（ユーザーエリア）は，32KRAMシステムで8000H～F37FH，16KRAMシステムでC000H～F37FHということになります。

それならユーザーはユーザーエリアのどこにでもプログラムを書き込めるかというと，そうではありません。MSX-BASICインタープリタはユーザーエリアを以下のように分けて管理しています。

①テキスト・エリア②変数領域（単純変数，配列変数）③フリーエリア④スタック・エリア⑤文字列領域⑥ファイルコントロール・ブロック

これらの領域がメモリ内のどこから始まっているかという情報はワーク・エリアに記録されています。ワーク・エリアのどの番地に記録されているか，初期値がいくつかということについては**表5-1**を参照してください（ただし初期値は32KRAMのとき）。では，それぞれの領域が使われる目的，領域内での表現（内部表現）などについて以下に解説しましょう。

表5-1　初期値は32Kシステム

領域の名	開始番地		終了番地	
	格納されているワーク・エリア	初期値	格納されているワーク・エリア	初期値
テキスト・エリア	F676H(TXTTAB)	8001H(変化しない)	(F6C2H)－1 →VARTAB	8003H(TXTTAB＋2)
単純変数領域	F6C2H(VARTAB)	8003H	(F6C4H)－1 →ARYTAB	—
配列変数領域	F6C4H(ARYTAB)	8003H	(F6C6H)－1 →STREND	—
フリーエリア	F6C6H(STREND)	8004H	—	—
スタック・エリア	そのときの (SP)	—	F674H(STKTOP)	F0A0H
文字列領域	(F674H)＋1 →STKTOP	F0A1H	F672H(MEMSIZ)	F168H
ファイルコントロール・ブロック	(F672H) ＋1 →MEMSIZ	F169H	FC4AH(HIMEM)	F380H
空き文字列領域の先頭	F69BH(FRETOP)	F168H		
MAXFILES の値	F85FH(MAXFIL)	1		

②各領域の内容

●テキスト・エリア(プログラム領域)

BASICで書かれたプログラムのテキストを格納します。ここに格納されるのは行番号が先頭についたプログラムだけで，ダイレクト・モードにおける命令は格納されません。しかもテキストは文字そのままではなく，「中間コード」といわれる特別な形式に変換されています。テキストを中間コードによって表現するのはメモリの節約と解読時間の短縮化を目的としており，MSXに限らずほとんどのBASICでも採用されています。

テキスト・エリアの内容を見てみましょう。たとえば，

```
10  INPUT  A
20  PRINT  A*3
30  END
```

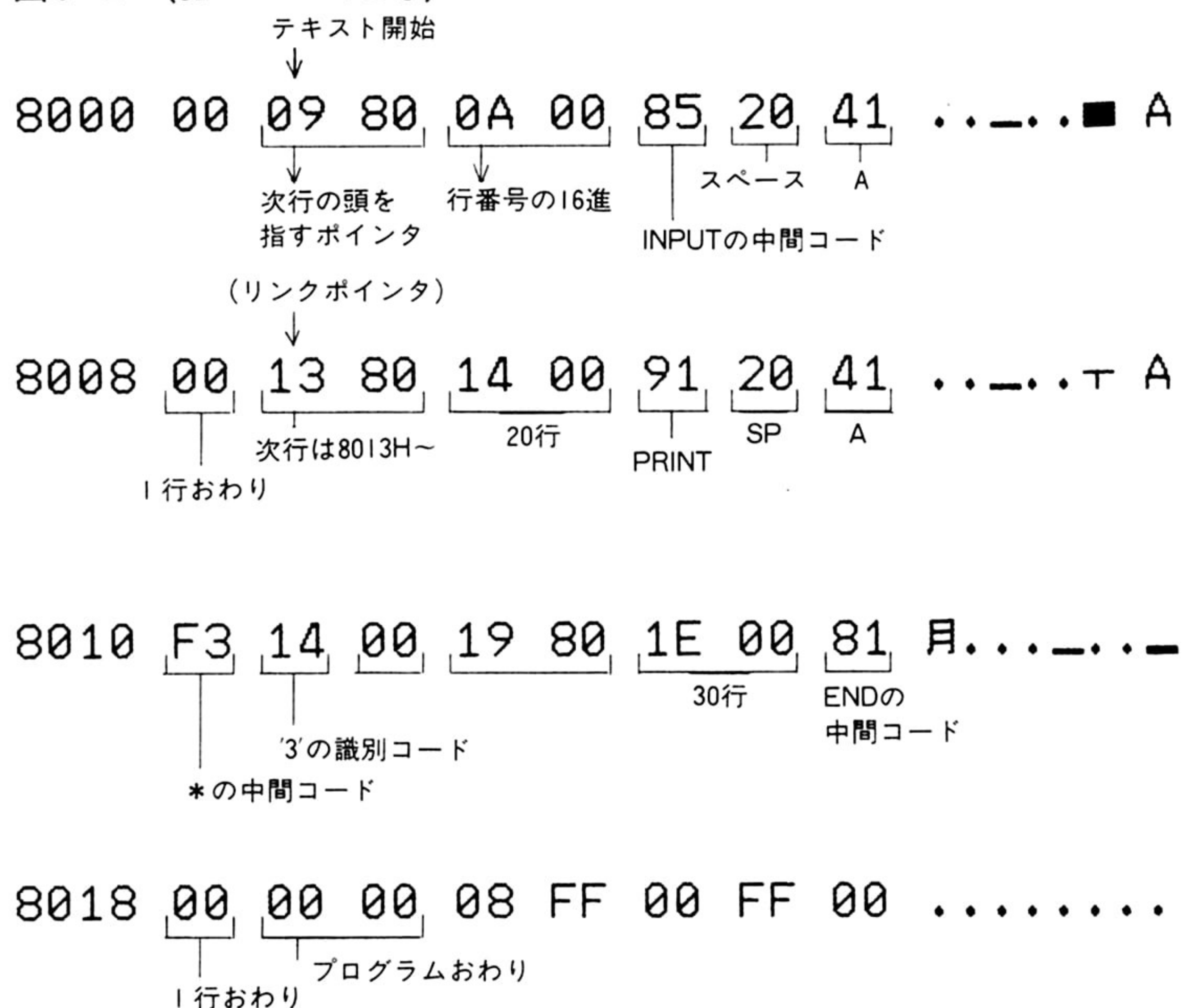
というプログラムをリセット直後に入力したとき，テキスト・エリアの内容をダンプしたのが**図5-2**です。32KRAMのときテキスト・エリアは8000H番地から始まります。8000H番地はいつもゼロで，8001H番地からテキストが入っています。8001～2H番地が「リンク・ポインタ」といわれるもので，次の行が始まる番地を指しています。つまりここでは20行の始まる番地が8009H番地ということです。

図5-2 (32KRAMのとき)

```
8000 00 09 80 0A 00 85 20 41  ..−..■ A
```
(テキスト開始 → 09 80：次行の頭を指すポインタ(リンクポインタ)／0A 00：行番号の16進／85：INPUTの中間コード／20：スペース／41：A)

```
8008 00 13 80 14 00 91 20 41  ..−..┬ A
```
(00：1行おわり／13 80：次行は8013H～／14 00：20行／91：PRINT／20：SP／41：A)

```
8010 F3 14 00 19 80 1E 00 81  月...−..−
```
(F3：*の中間コード／14：'3'の識別コード／1E 00：30行／81：ENDの中間コード)

```
8018 00 00 00 08 FF 00 FF 00  ........
```
(00：1行おわり／00 00：プログラムおわり)

8003～4H番地が行番号です。16進数表現で上位と下位が逆転していることに注意してください。2バイトですから表現できる行番号はFFFFH＝65535までということになります。ここでは0AH＝10です。

8005H番地がINPUT命令の中間コード85Hです。ここで中間コードをすべて紹介することはしませんが，演算子も含めた各命令ひとつひとつにほぼ1～2バイトが割り当てられています。

8006～7H番地はINPUT命令に続く空白と変数AとをASCIIコードにしてあるのです。

8008H番地のゼロは，行の終わりを示しています。もしこのあともゼロが2つ続けば（つまりリンク・ポインタがゼロなら），プログラム自体の終わりということになります。

PRINT命令の中間コードは91H，＊はF3Hだということがわかりますね。8011H番地の14Hは数値の識別コードといわれるもので，プログラム中の数値を変数や行番号と混同しないようにするための目印です。10進数1桁の「3」は識別コードで14Hというわけです。

また，GOTO，GOSUB，THENなどの命令に伴う行番号は打ち込んだ直後と実行後とで内部表現がちがいます。**図 5-3** は，前のリストの30行を

30 GOTO 10

に変えて打ち込んだ直後の内部表現です。8017H番地の89HがGOTO命令の中間コード，8018Hは空白です。8019H番地の0EHは識別コードのひとつで,「以下の2バイトはジャンプ先の行番号を16進数表現したものだよ」ということを示しています。801A～BHは0AHで，確かに飛び先の行番号です。

図5－3

```
8000 00 09 80 0A 00 85 20 41 ..＿..■ A
8008 00 13 80 14 00 91 20 41 ..＿..┬ A
8010 F3 14 00 1D 80 1E 00 89 月...＿..▌
8018 20 0E 0A 00 00 00 00 00  .......
8020 FF 00 FF 00 FF 00 FF 00 ........
8028 FF 00 FF 00 FF 00 FF 00 ........
```

実行後の内部表現がP140の**図 5-4** です。ここでは8019Hの識別コードが0DHに，そのあとの2バイトが8000Hに変化しています。識別コード0DHは「以下の2バイトはジャンプ先の実アドレスだよ」という意味で，確かに8000H番地は10行の1バイト前ですね。このようにして実行の高速化をはかっているのです。

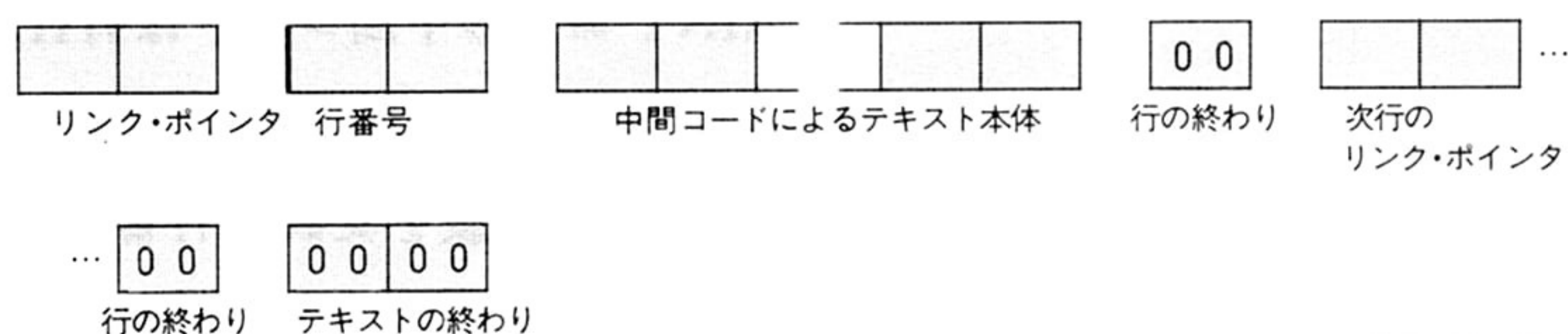
テキスト・エリア内の構造を図5-5に図示しておきます。

図5-4

```
8000 00 09 80 0A 00 85 20 41 ..▁..■ A
8008 00 13 80 14 00 91 20 41 ..▁..┬ A
8010 F3 14 00 1D 80 1E 00 89 月....▁..▌
8018 20 0D 00 80 00 00 00 08  ..▁....
8020 41 00 41 30 00 00 00 00 A.A0....
8028 00 00 00 00 FF 00 FF 00 ........
```

図5-5

リンク・ポインタ	行番号	中間コードによるテキスト本体	00 行の終わり	次行のリンク・ポインタ …

… 00 行の終わり	00 00 テキストの終わり

●単純変数領域，配列変数領域

単純変数，配列変数などの変数が使われたときその変数名と値とを対応づけて登録しておくエリアです。変数名にはその型もいっしょに登録されます。MSX-BASICの変数の型には，

・数値型－整数型（変数名に％をつける）
　　　　－単精度型（同じく！をつける）
　　　　－倍精度型（同じく＃をつけるか，何もつけない）
・文字型（変数名に＄をつける）

の4通りがあって，それぞれ内部表現や占有バイト数がちがっています。もちろん，単純変数と配列変数とでもバイト数がちがいます。

配列変数については1つだけ注意すべきことがあります。それは，通常，配列変数はDIM命令で添字の最大値を宣言してから使うのに，添字が10以下の大きさならDIM命令で宣言しなくてもエラーにならないということです。このような配列変数は，使われたときに自動的に添字0～10まで11個分の領域が確保されます。

ここでは数値型変数と文字型変数とに分けて解説します。

〈数値型変数〉

リスト5-1のプログラムをリセット直後に入力・RUNさせたときの変数領域を見たのが**図5-6**です。まずワーク・エリアのF6C0H～をのぞいて単純・配列の各変数領域がどこを占めているかを調べました。単純変数領域が8036～56H，配列変数領域が8057～B6H

リスト5-1

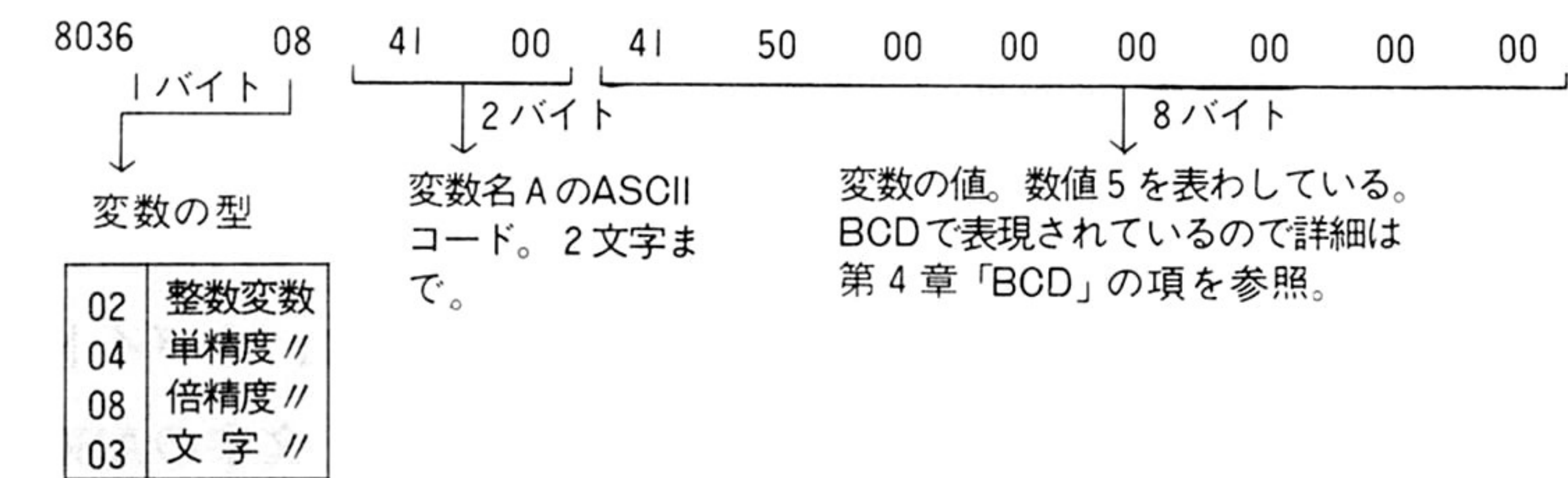
```
10 INPUT A
20 INPUT B
30 C=A*B
40 D(0)=C
50 PRINT C
60 END
```

であることがわかります。

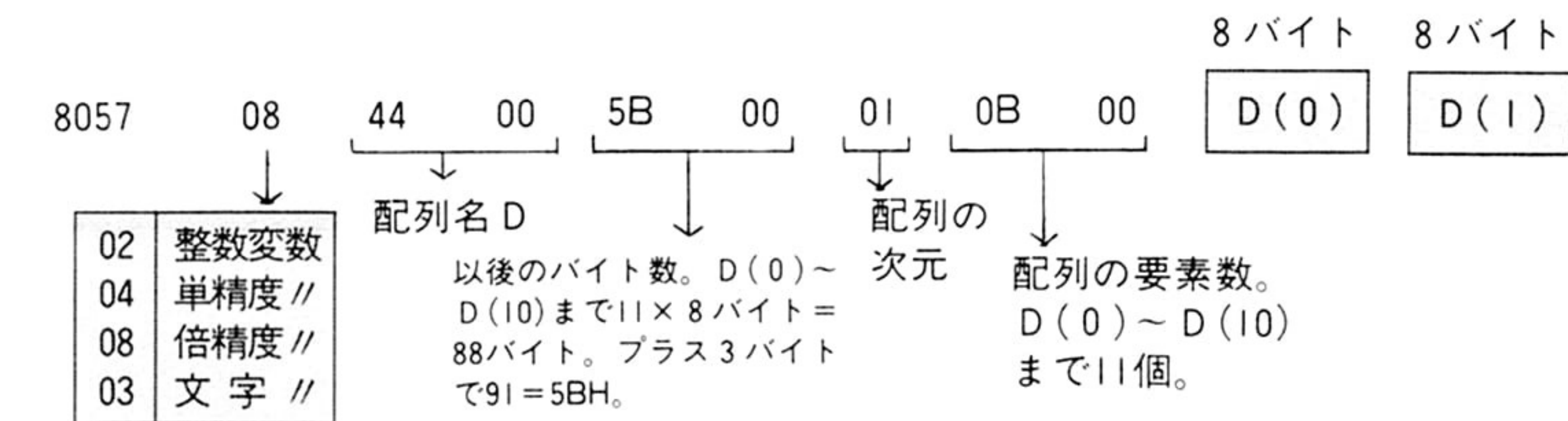
図5－6

```
F6C0 33 80 36 80 57 80 B7 80 3_6_W_ｷ_



8030 3C 00 81 00 00 00 08 41 <.▬....A
8038 00 41 50 00 00 00 00 00 .AP.....
8040 00 08 42 00 41 60 00 00 ..B.A`..
8048 00 00 00 00 08 43 00 42 .....C.B
8050 30 00 00 00 00 00 00 08 0.......
8058 44 00 5B 00 01 0B 00 42 D.[....B
8060 30 00 00 00 00 00 00 00 0.......
8068 00 00 00 00 00 00 00 00 ........
8070 00 00 00 00 00 00 00 00 ........
8078 00 00 00 00 00 00 00 00 ........
8080 00 00 00 00 00 00 00 00 ........
8088 00 00 00 00 00 00 00 00 ........
8090 00 00 00 00 00 00 00 00 ........
8098 00 00 00 00 00 00 00 00 ........
80A0 00 00 00 00 00 00 00 00 ........
80A8 00 00 00 00 00 00 00 00 ........
80B0 00 00 00 00 00 00 00 00 ........
80B8 FF 00 FF 00 FF 00 FF 00 ........
```

そこで続けて8030～BFHまでをダンプしてみました。このうち8036～40Hは次のように考えることができます。

02	整数変数
04	単精度〃
08	倍精度〃
03	文字〃

ここでもし変数Aが単精度（A！）なら変数の型は4になり，変数の値は4バイトを占めることになります。変数の型の値は，値部のバイト数と考えてもよいのです。

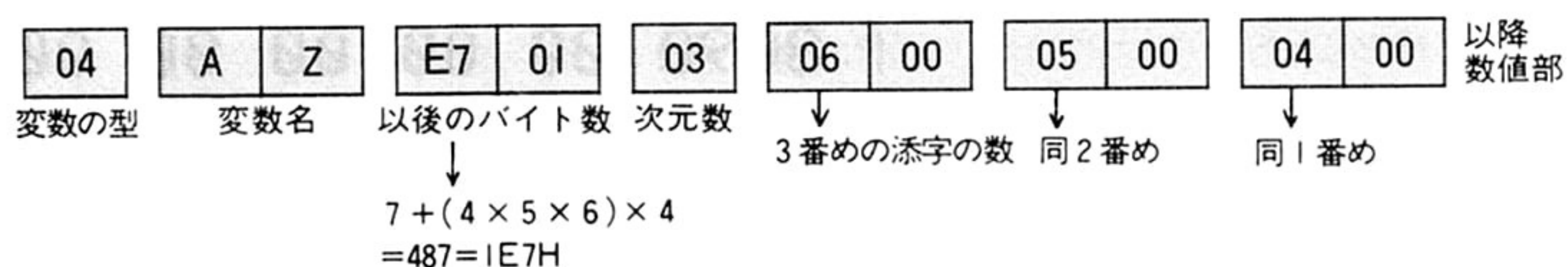
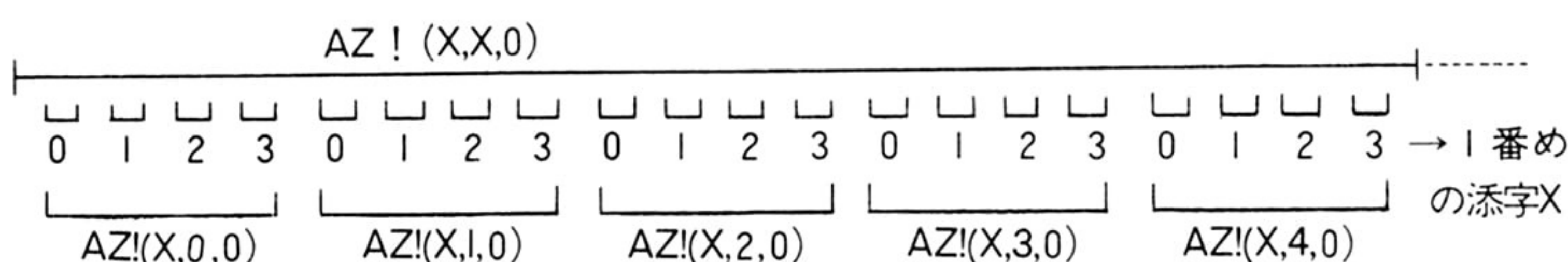
8057Hからの配列変数領域はどうでしょう。ここではDという名でD（0）～D（10）までを使い，D(0）だけに値を入れましたね。

02	整数変数
04	単精度〃
08	倍精度〃
03	文字〃

上のようになっています。Dは倍精度ですから値部は8バイトずつ格納されています。配列の次元数は255（16進数でFFH）まで許される規則になっていますが，実際は入力時の1行が255文字まで，しかも次元数を増やせば配列領域の必要バイト数が級数的に増えるため，せいぜい3次元までしか実用にならないでしょう。

単純変数・配列変数のうち，数値型変数の格納のされかたを図5-7に示します。

図5-7（単純変数）

	変数の型	変数名		数値部		備考
整数変数	02	1文字めのASCIIコード	2文字め（ないときは00）	2バイト	—	計11バイト
単精度変数	04	〃	〃	4バイト	—	計7バイト
倍精度変数	08	〃	〃	8バイト		計5バイト

図5-7（配列変数） AZ!(3，4，5）のとき

〈文字型変数〉

数値変数は精度によって内部表現で占める値部の大きさ(バイト数)が一意に決まり，2，4，8バイトのいずれかです。ところが文字列を値とする文字変数は，文字のASCIIコードをその長さだけ持つとすると，文字列の長さによって値部の大きさが変わってきます。MSX-BASICは文字列の長さを「255文字以下」と規定しているだけなのです。

そこでMSX-BASICの内部表現では，変数領域に文字データそのものを持ち込まないしくみになっています。変数領域内の文字単純変数は次のような形で格納されています。

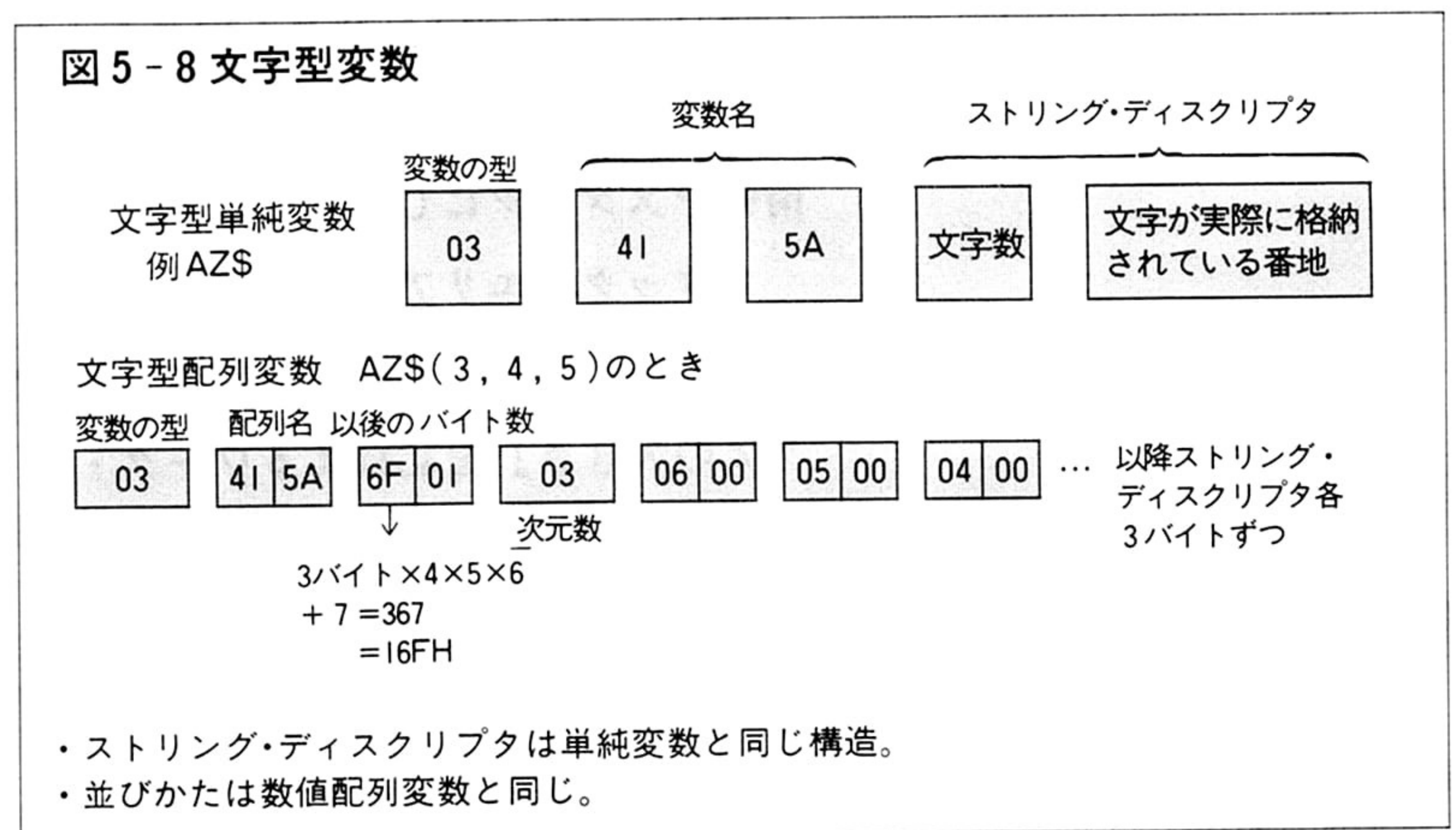
文字単純変数の内部表現による大きさは6バイトで，最初の1バイトに変数の型（文字型は3），つづいて変数名2バイト，最後に「ストリング・ディスクリプタ」と呼ばれるものになっています。このストリング・ディスクリプタこそ，「文字列そのものを持ち込まないしくみ」なのです。

ストリング・ディスクリプタの先頭1バイトは文字列本体の長さを示し，あとの2バイトは文字列本体が実際に格納されているメモリのアドレスを示しています。このアドレスは原則的に文字列領域という特別な場所を指しています。文字列領域内のしくみについては次頁の⑤を見てください。

文字型配列変数についても基本は同じで，数値型配列変数における値部に各3バイトのストリング・ディスクリプタが入るようになっています。

文字型単純変数，配列変数のそれぞれについて，格納のされかたを**図 5-8** に示します。

図5－8 文字型変数

・ストリング・ディスクリプタは単純変数と同じ構造。
・並びかたは数値配列変数と同じ。

③フリーエリア

フリーエリアはユーザーエリアのうちでまだ用途が決まっていない領域のことです。ただしここもBASICインタープリタの管理内なので，空いているからといってユーザーが勝手に使うことはできません。BASICインタープリタは人間が使っているのも知らずに変数領域や後述のスタック・エリアを伸ばしてくる恐れがあるからです。人間が独自のエリアを使いたいときはそのための宣言をせねばなりません（CLEAR命令の項参照)。

BASICにはFREという関数があり，これによってフリーエリアの大きさ（バイト数）を知ることができます。1度ダイレクトに

PRINT　FRE（0）↵

とやってみてください。そのときのフリーエリアの大きさが10進数で表示されます。変数やプログラム領域が使われたり，スタックが伸びたりするとフリーエリアはどんどん減っていきます。

④スタック・エリア

ここはBASICインタープリタが使うスタックのエリアです。マシン語におけるCALL命令と同じように，BASICのGOSUB命令やFOR命令を実行するときにもBASICインタープリタは戻り番地などの情報をスタックにしまうのです。

スタック・エリアは常に幅が増減しているので，ある一瞬の大きさをはかってよしとするわけにはいきません。ですから,「スタック・エリアの大きさ」を記録するワーク・エリアもありません。

⑤文字列領域

文字を値とする変数や定数がその値を格納する特別な領域がこの文字列領域です。

文字列領域はリセット時に200バイトあり，この大きさは後述のCLEAR命令で変えることもできます。文字型変数のストリング・ディスクリプタのポインタはたいていこの領域内に格納された文字列本体の1文字めを指しています。ところが，プログラム中の代入文（A$＝“ABC”など)やDATA命令のデータ，PRINT命令で使われた文字列は文字列領域にも移されず，ストリング・ディスクリプタのポインタもテキスト中を指しています。つまり，プログラム中で文字列に「文字列演算」をほどこさない限り文字列領域が使われないというわけです。こうした文字列（とくに文字定数）は1度限りの使い捨てだからです。

また，文字列領域に文字列を格納するときは領域の後尾から順に，

後ろから前へと詰められます。もし文字列領域がいっぱいになったら，文字列領域にすでに詰められている文字列を調べ，もう使われていない文字列を削除していきます（この動作をガベージ・コレクションという)。ガベージ・コレクションをしてもなお文字列領域に空きができないときには“Out of string space”エラーとなります。

文字列領域の使われかたの例として，**リスト 5-2** を用意しました。リセット後このリストを入力・RUN してみてください。このプログラムは，キー入力された文字が A$ という配列変数の中にあればそれ以降の文字を，なければないと表示します。

RUN させてから“YAMAHA”とキー入力して CTRL-STOP し，単純変数領域，配列変数領域，文字列領域をダンプすると下のようになっています。

リスト 5-2

```
100 M$="ﾅﾏｴ ﾊ"
110 FOR I=0 TO 5
120 READ A$(I)
130 NEXT I
140 '
150 INPUT B$
160 FOR J=0 TO 5
170 S=INSTR(A$(J),B$)
180 IF S=0 THEN NEXT ELSE 200
190 PRINT M$+" ｱﾘﾏｾﾝ":GOTO 150
200 PRINT M$+MID$(A$(J),LEN(B$)+1)
210 GOTO 150
220 '
230 DATA "YAMAHA YIS503"
240 DATA "NATIONAL KING-KONG"
250 DATA "SONY HITBIT"
260 DATA "TOSHIBA PASOPIA-IQ"
270 DATA "PIONEER PALCOM"
280 DATA "SANYO WAVY"
```

```
814A 03 4D 00 05 09 80 08 49 .M...＿.I
8152 00 41 60 00 00 00 00 00 .A`.....
815A 00 03 42 00 06 EE D1 08 ..B../ﾑ.
8162 4A 00 00 00 00 00 00 00 J.......
816A 00 00 08 53 00 41 10 00 ...S.A..
8172 00 00 00 00 00 03 41 00 ......A.
817A 24 00 01 0B 00 0D C5 80 $.....ﾅ＿
8182 12 DB 80 0B F6 80 12 0A .□＿.ｶ＿..
818A 81 0E 25 81 0A 3C 81 00 ＿.%＿.<＿.
8192 00 00 00 00 00 00 00 00 ........
819A 00 00 00 00 00 00       ......

D1D0 FF 00 FF 00 FF 00 FF 00 ........
D1D8 FF 00 FF C5 CF B4 20 CA ...ﾅﾏｴ ﾊ
D1E0 20 59 49 53 35 30 33 20  YIS503 
D1E8 59 49 53 35 30 33 59 41 YIS503YA
D1F0 4D 41 48 41 FF F7 D1 00 MAHA.秒ﾑ.
```

各文字変数について見てみると，

変数名	格納アドレス	ストリング・ディスクリプタ			実文字列の場所
		アドレス	文字数	ポインタ	
M$	814AH	814DH	5	8009H	テキスト
B$	815BH	815EH	6	D1EEH	文字列領域
A$(0)	8177H	817FH	13	80C5H	テキスト
A$(1)	－	8182H	18	80DBH	〃
A$(2)	－	8185H	11	80F6H	〃
A$(3)	－	8188H	18	810AH	〃
A$(4)	－	818BH	14	8125H	〃
A$(5)	－	818EH	10	813CH	〃

となります。このうちB$を除いては文字列本体がテキスト・エリアの中にあるままですね。B$はプログラムの200行で文字列演算しているために文字列領域に移されたのです。

⑥ファイルコントロール・ブロック

ファイルコントロール・ブロックはMSXが外部と入出力する際にバッファとして使われるエリアです。MSXがOPEN命令でオープンできるファイルはCAS：（カセット・テープ），CRT：（テキスト画面），GRP：（グラフィック画面），LPT：（プリンタ）の4つです。ただし，プログラム・ファイル（プログラム自体もファイル＝意味のある情報の集合＝と考える）の入出力（ロード，セーブのこと）にはファイルをオープンする必要がありません。

同一プログラム上でファイルをいくつオープンできるかは，ファイルコントロール・ブロックがどれだけ確保されているかによります。ファイル数の最大値を指定するBASIC命令がMAXFILESで，たとえばMAXFILES＝2とするとプログラム・ファイル以外に2つのファイルをオープンできます。＝0ならプログラム・ファイル以外の入出力はできません。リセット時は＝1となっています。

ファイルコントロール・ブロックの大きさは，ファイル1つにつき267バイトです。それに加えてプログラム・ファイル用に常時1ファイル分が確保されているので，リセット時には2ファイル分のバッファがとられていることになります（**表5-1** 参照）。ちなみにMAXFILES数の現在値もワーク・エリア上に記録されています。

2.CLEAR命令

BASICのCLEAR命令は，いわば初期化命令です。その動作はおよそ次の通りです。

①すべての数値変数を0に，文字変数を“ ”（ヌル，空文字列）にする。

②DEF命令で定義された情報（DEF FN，DEFINT，DEFSNG，D-

EFDBL, DEFSTR, DEFUSR）をすべて無効にする。

③ オープン中のファイルをすべてクローズする。

④ 文字列領域の大きさ，BASICのユーザーエリアの上限を指定する。

⑤ NEXT命令に対応するFOR命令，RETURN命令に対応するGOSUB命令などを忘れてしまう（スタックの初期化）。

つまり，CLEAR命令を実行するとBASICはほとんど何もかも忘れてしまい，メモリの使いかたまで変わるのです。

ここで特に取り上げるのは④についてです。CLEAR命令には2つのパラメータがあり，

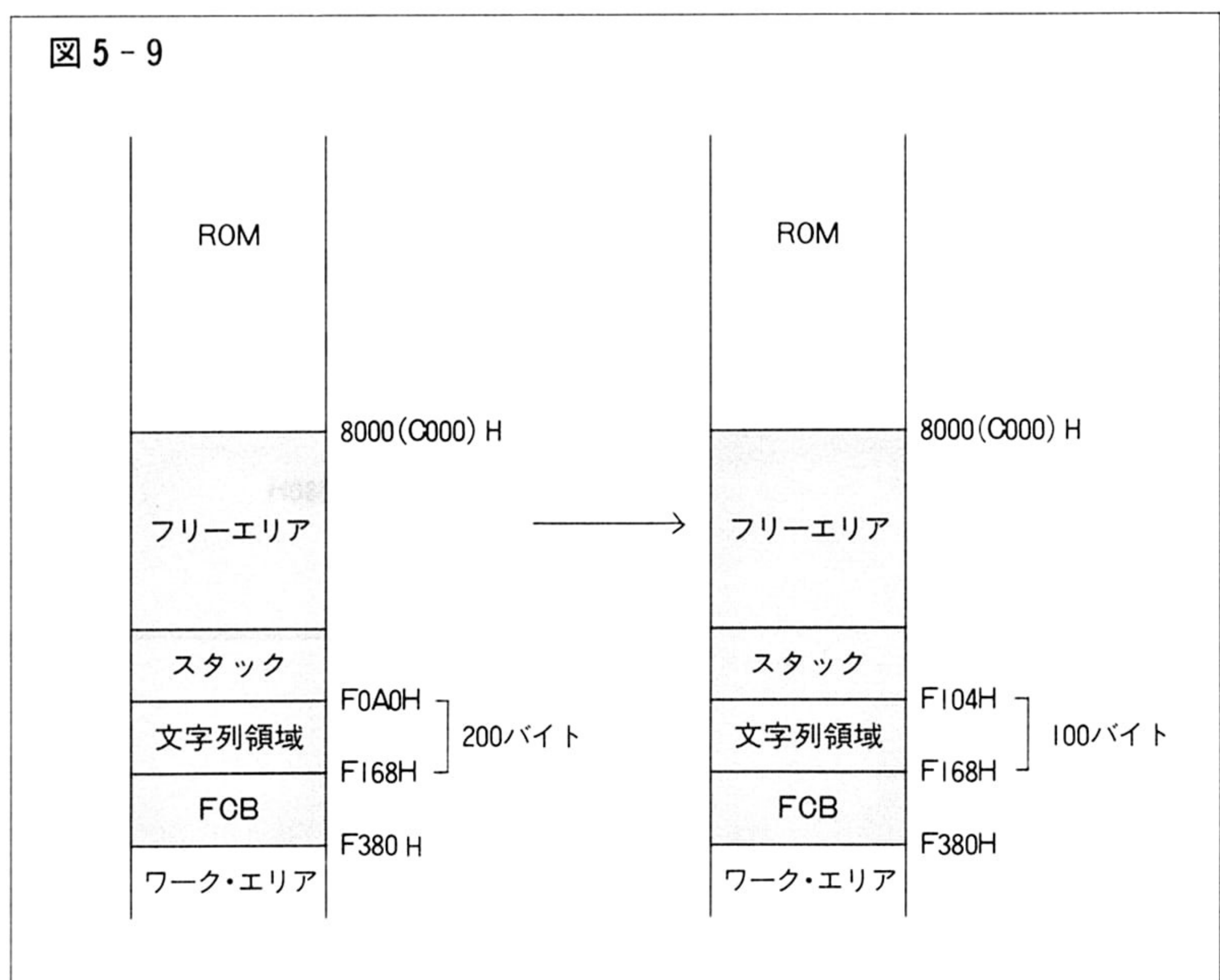
CLEAR 〈文字列領域の大きさ〉,〈ユーザーエリアの上限〉

となっています。このどちらも省略でき，省略したときはメモリ・マップを変更しません。ただし,「ユーザーエリアの上限」を指定するときは,「文字列領域の大きさ」を省略することができません（CLEAR ,&H…ということはできない)。初期値は文字列領域の大きさ＝200バイト，ユーザーエリアの上限＝F380Hとなっています。

ためしにリセット（または電源OFF）後，ダイレクトに

```
CLEAR 100↵
```

としてみたときのメモリ・マップの変化を**図5-9**に示します。F168H-F104H＝100ですね。

図5-9

次にユーザーエリアの上限を変えるということについて解説しましょう。**図5-1** をもう1度見てください。BASICのROMに続くテキスト・エリアからワーク・エリアの直前までがユーザーエリアと呼ばれていますね。このユーザーエリアを図の上方向に縮めて，ファイルコントロール・ブロック・エリアとの間にすき間をつくってしまおうというのが「ユーザーエリアの上限を変える」ことの主旨です。具体的には，たとえばリセット直後に

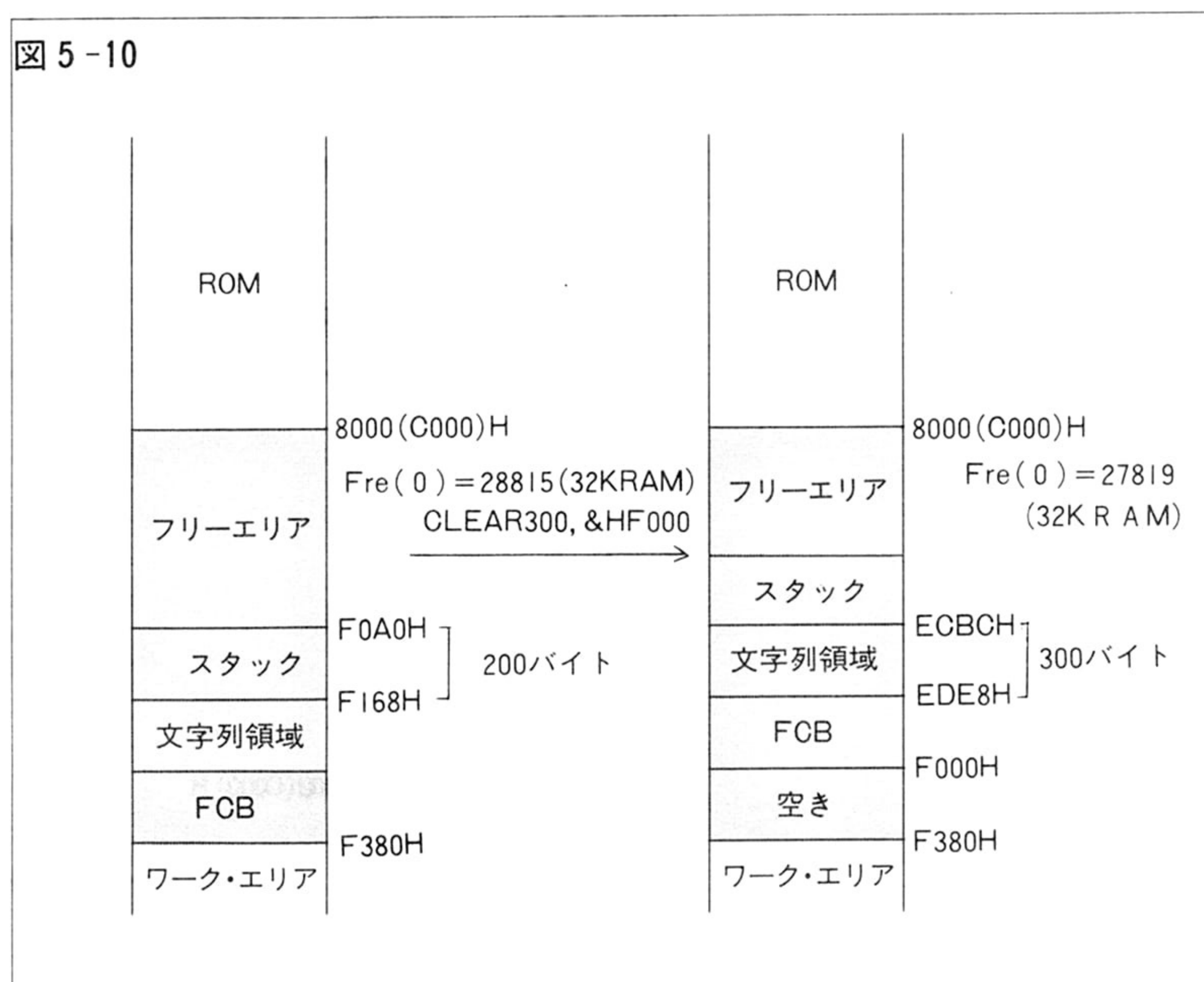
CLEAR　300，&HF000↵

とすれば，ファイルコントロール・ブロックの終端がF380H→F000Hに変わり，ワーク・エリアの先頭との間にF001H～F37FHまで895バイトのすき間ができるわけです。このすき間はBASICの管理外なので，ユーザーが自由に使うことができます（**図5-10**）。

このエリアはユーザーが自分のマシン語プログラムを書き込むのに使います。よくBASIC＋マシン語のゲーム・プログラムでBASICリストの先頭にCLEAR命令を書いているのはこのためです。

図5-10

ROM
8000(C000)H
フリーエリア
Fre(0)=28815(32KRAM)
CLEAR300, &HF000 →
F0A0H
スタック
200バイト
F168H
文字列領域
FCB
F380H
ワーク・エリア

ROM
8000(C000)H
フリーエリア
Fre(0)=27819
(32KRAM)
スタック
ECBCH
文字列領域
300バイト
EDE8H
FCB
F000H
空き
F380H
ワーク・エリア

3.USR関数

USR関数は0～9までの10個の番号をつけて区別した，メモリ内のマシン語ルーチンを呼び出す命令です。呼び出す前にDEFUSR命令でその番号の実行開始番地を設定しておかなくてはなりません。

また，USR命令はもともと単なるマシン語ルーチンの呼び出しのためというよりは「マシン語で書いた関数（ユーザー関数）を呼び出す」ためのものであるため，パラメータを引き渡す「関数」の形をとっています。パラメータはUSR関数のカッコ内に書き，呼び出されたマシン語ルーチンへその値が渡されます。このパラメータは省略できないため，パラメータが必要ないマシン語ルーチンを呼ぶときもカッコ内にダミー（かたちだけ）のパラメータを書かねばなりません。マシン語ルーチンを実行した結果はUSR関数の値（A＝USR…のA）として戻ってきます。

DEFUSRで定義したマシン語ルーチンの開始番地は，ワーク・エリアのF39AH（USRTAB）からの20バイトにユーザー番号順におさめられています。

```
F39A 5A 47 5A 47 5A 47 5A 47 ZGZGZGZG
F3A2 5A 47 5A 47 5A 47 5A 47 ZGZGZGZG
F3AA 5A 47 5A 47             ZGZG
```

初期値は10個とも475AH番地で，これはIllegal function callエラーを出すルーチンです。

DEFUSR　5＝&HAABB↵

としたあと，同じエリアをダンプしたのが下のリストです。

```
F39A 5A 47 5A 47 5A 47 5A 47 ZGZGZGZG
F3A2 5A 47 BB AA 5A 47 5A 47 ZGｻｪZGZG
F3AA 5A 47 5A 47             ZGZG
```

F39AH番地をUSR番号0として数えるとF3A4H番地がUSR番号5ですね。AABBHが入っています。

DEFUSRで開始番地を定義した後，USRで呼び出します。USR関数の引数は数値型でも文字型でも可です。USR関数を実行するとまず引数の型が調べられます。引数が数値型であればその精度も調べられ，各レジスタに引数の型やアドレスが入れられてからユーザーのマシン語ルーチンをコールします。コールですからユーザールーチンの最後にはRET命令がなくてはなりません。引数の型によってセット

されるレジスタと値は図5-11の通りです。

図5-11

①数値型

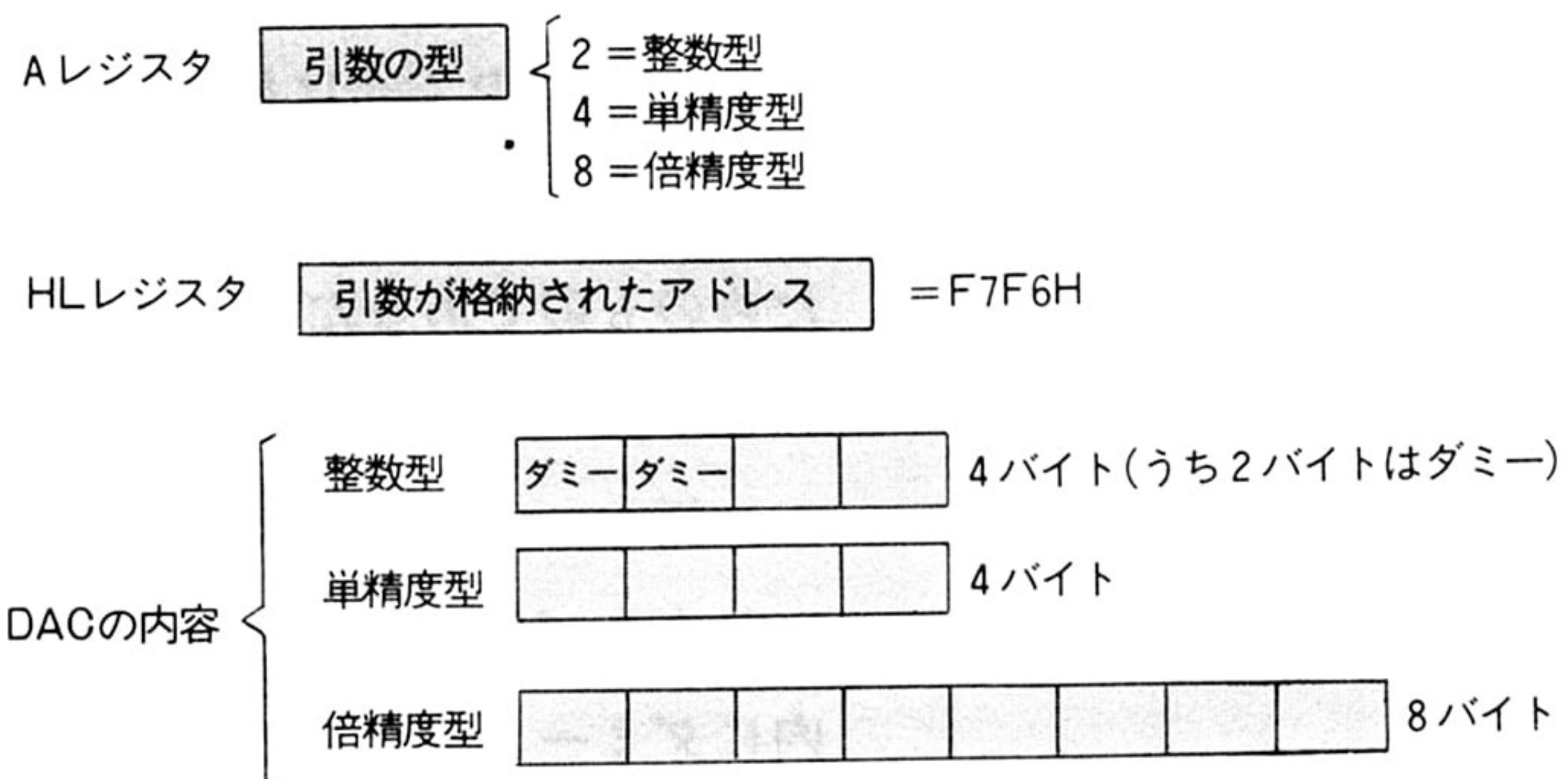
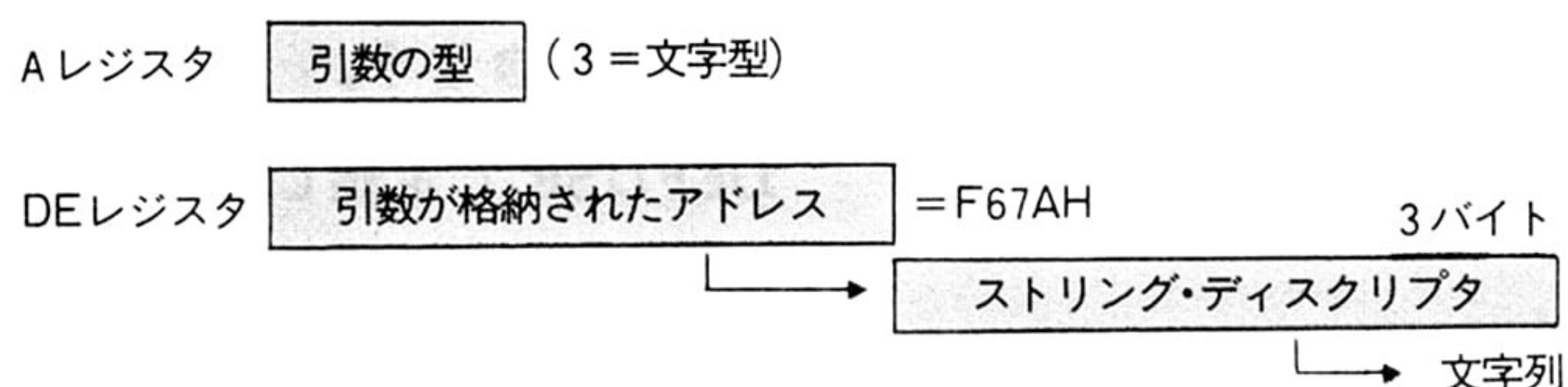
Aレジスタ　引数の型　2＝整数型 / 4＝単精度型 / 8＝倍精度型

HLレジスタ　引数が格納されたアドレス　＝F7F6H

DACの内容
- 整数型　ダミー｜ダミー｜｜　4バイト(うち2バイトはダミー)
- 単精度型　｜｜｜　4バイト
- 倍精度型　｜｜｜｜｜｜｜　8バイト

②文字型

Aレジスタ　引数の型　(3＝文字型)

DEレジスタ　引数が格納されたアドレス　＝F67AH
→ ストリング・ディスクリプタ（3バイト）
→ 文字列

特に引数が数値型の場合，引数の値そのものはワーク・エリアのDAC（Decimal ACcumulator＝F7F6H）にコピーされます。つまり，引数が数値型のときHLレジスタの内容はいつもF7F6Hというわけです。DACの内容は変数における数値の内部表現にほぼ準じていますが，引数が整数型であるときに限ってDACの先頭から3バイト，4バイトめに値が入れられ，1～2バイトめは使われません。

引数が文字型であったときはDEレジスタにストリング・ディスクリプタのある番地が入ります。ストリング・ディスクリプタはいつもワーク・エリアのF67AH（TEMPST）からつくられ，その構造は文字型変数のそれと同じです。

ユーザーのマシン語ルーチンで加工した引数をBASICに返したいときは，引数の格納されたメモリ(数値型ならDAC，文字型ならストリング・ディスクリプタの指す番地)をユーザールーチン内で直接書きかえます。このとき注意しなければいけないのは文字型の引数のときです。文字列本体はテキスト・エリアか文字列領域に置かれているわけで，もしユーザールーチン内で文字列の長さを変えるような操作をした場合，隣接するテキストや文字列領域の内容を書きかえることになるのです。テキスト・エリアを書きかえたらどうなるかはわかりますね。文字列領域でも他の文字列を書きかえてしまう恐れは十分にあります。

数値型も文字型も，引数の型自体をユーザールーチン内で変えることはできません。

リスト5-3 と **4**は，それぞれ数値型，文字型変数を引数にしたときのUSR関数の動作を示すサンプル・プログラムです。**リスト5-3**は入力された4桁の文字をアドレスを表現した16進数とみてマシン語ルーチンに渡し，マシン語ルーチン側ではそのアドレス以降128バイトをダンプするものです。ダンプ・ルーチンは第3章で作成したものを使いました。**リスト5-3（その2）**のアセンブル・リストをアセンブルしてメモリ（D400H番地～）に入れておけば，あとは（その1）のリストを入力・RUNするかダイレクトにUSR命令を実行するだけでダンプされます。

リスト5-4 は，引数にした文字列を逆順に表示するものです。

リスト5-3（その1）

```
100 'SAMPLE (USR)
110 '
120 DEFUSR=&HD400
130 INPUT A$
140 A=USR(VAL("&H"+A$))
150 END
```

リスト5-3（その2）

```
  MSX Self Assembler  Rev 1.0    PAGE    1


1000:   D400                     ORG     0D400H
1010:                 ;
1020:   00A2 =        CHPUT      EQU     00A2H
1030:   00B7 =        BREAKX     EQU     00B7H
1040:                 ;
1050:   000D =        CR         EQU     0DH
1060:   000A =        LF         EQU     0AH
1070:                 ;
1080:   D400 23                  INC     HL
1090:   D401 23                  INC     HL
1100:   D402 5E                  LD      E,(HL)
1110:   D403 23                  INC     HL
1120:   D404 66                  LD      H,(HL)
1130:   D405 6B                  LD      L,E
1140:   D406 E5                  PUSH    HL
1150:   D407 017F00              LD      BC,127
1160:   D40A 09                  ADD     HL,BC
1170:   D40B D1                  POP     DE
```

```
1180:                      ;
1190:  D40C                DUMP:
1200:  D40C E5                       PUSH    HL
1210:  D40D D5                       PUSH    DE
1220:  D40E C5                       PUSH    BC
1230:  D40F                DUMP2:
1240:  D40F CD29D4                   CALL    ADRP
1250:  D412 0608                     LD      B,8
1260:  D414                ONELIN:
1270:  D414 CD47D4                   CALL    DATAP
1280:  D417 13                       INC     DE
1290:  D418 E7                       RST     20H
1300:  D419 380A                     JR      C,EXIT
1310:  D41B 10F7                     DJNZ    ONELIN
1320:                      ;
1330:  D41D CD5BD4                   CALL    CRLF
1340:  D420 CDB700                   CALL    BREAKX
1350:  D423 30EA                     JR      NC,DUMP2
1360:                      ;
1370:  D425                EXIT:
1380:  D425 C1                       POP     BC
1390:  D426 D1                       POP     DE
1400:  D427 E1                       POP     HL
1410:  D428 C9                       RET
1420:                      ;
1430:                      ;
1440:  D429                ADRP:
1450:  D429 7A                       LD      A,D
1460:  D42A CD66D4                   CALL    TOASC
1470:  D42D 78                       LD      A,B
1480:  D42E CDA200                   CALL    CHPUT
1490:  D431 79                       LD      A,C
1500:  D432 CDA200                   CALL    CHPUT
1510:  D435 7B                       LD      A,E
1520:  D436 CD66D4                   CALL    TOASC
1530:  D439 78                       LD      A,B
1540:  D43A CDA200                   CALL    CHPUT
1550:  D43D 79                       LD      A,C
1560:  D43E CDA200                   CALL    CHPUT
1570:  D441 3E20                     LD      A,' '
1580:  D443 CDA200                   CALL    CHPUT
1590:  D446 C9                       RET
1600:                      ;
1610:  D447                DATAP:
1620:  D447 C5                       PUSH    BC
1630:  D448 1A                       LD      A,(DE
1640:  D449 CD66D4                   CALL    TOASC
1650:  D44C 78                       LD      A,B
```

```
1660:    D44D CDA200                   CALL     CHPUT
1670:    D450 79                       LD       A,C
1680:    D451 CDA200                   CALL     CHPUT
1690:    D454 3E20                     LD       A,' '
1700:    D456 CDA200                   CALL     CHPUT
1710:    D459 C1                       POP      BC
1720:    D45A C9                       RET
1730:                     ;
1740:    D45B             CRLF:
1750:    D45B 3E0D                     LD       A,CR
1760:    D45D CDA200                   CALL     CHPUT
1770:    D460 3E0A                     LD       A,LF
1780:    D462 CDA200                   CALL     CHPUT
1790:    D465 C9                       RET
1800:                     ;
1810:    D466             TOASC:
1820:    D466 F5                       PUSH     AF
1830:    D467 E6F0                     AND      0F0H
1840:    D469 CD77D4                   CALL     ASC2
1850:    D46C 47                       LD       B,A
1860:    D46D F1                       POP      AF
1870:    D46E F5                       PUSH     AF
1880:    D46F E60F                     AND      0FH
1890:    D471 CD7FD4                   CALL     ASC3
1900:    D474 4F                       LD       C,A
1910:    D475 F1                       POP      AF
1920:    D476 C9                       RET
1930:                     ;
1940:    D477             ASC2:
1950:    D477 CB3F                     SRL      A
1960:    D479 CB3F                     SRL      A
1970:    D47B CB3F                     SRL      A
1980:    D47D CB3F                     SRL      A
1990:    D47F             ASC3:
2000:    D47F FE0A                     CP       0AH
2010:    D481 3802                     JR       C,SUUJI2
2020:    D483 C607                     ADD      A,7H
2030:    D485             SUUJI2:
2040:    D485 C630                     ADD      A,30H
2050:    D487 C9                       RET
```

リスト5-4

```
100 'SAMPLE 2
110 '
120 AD=&HD400
130 READ A$
140 IF A$="FF" THEN 190
150 POKE AD,VAL("&H"+A$)
160 AD=AD+1
170 GOTO 130
180 '
190 DEFUSR 1=&HD400
200 INPUT "ﾓｼﾞﾚﾂ ";CH$
210 IF CH$="" THEN 200
220 PRINT
230 DM$=USR 1(CH$)
240 END
250 '
260 'MACHINE LANGUAGE DATA
270 '
280 DATA EB,46,23,5E,23,56,EB,58
290 DATA 16,00,19,2B,7E,CD,A2,00
300 DATA 2B,10,F9,C9,FF
```

リスト5-4(データ部のアセンブル・リスト)

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    D400                     ORG      0D400H
110:
120:    00A2 =         CHPUT     EQU      00A2H
130:
140:    D400 EB                  EX       DE,HL
150:    D401 46                  LD       B,(HL)
160:    D402 23                  INC      HL
170:    D403 5E                  LD       E,(HL)
180:    D404 23                  INC      HL
190:    D405 56                  LD       D,(HL)
200:    D406 EB                  EX       DE,HL
210:    D407 58                  LD       E,B
220:    D408 1600                LD       D,0
230:    D40A 19                  ADD      HL,DE
240:    D40B 2B                  DEC      HL
250:    D40C 7E        LOOP:     LD       A,(HL)
260:    D40D CDA200              CALL     CHPUT
270:    D410 2B                  DEC      HL
280:    D411 10F9                DJNZ     LOOP
290:    D413 C9                  RET
```

4.VARPTR関数

VARPTR関数は，引数として与えられた変数の値部（文字型ならストリング・ディスクリプタ）の先頭アドレスを求めます。引数に＃を伴うファイル番号を与えると，ファイルコントロール・ブロック中のそのファイル番号のバッファ先頭アドレスを返します。

例としてBASICのダイレクト・モードでAB＝0↵としてから

```
PRINT  HEX$ (VARPTR (AB)) ↵
```

としてみてください。4桁の16進数が表示されましたね。モニタのDコマンドでその16進数をアドレスとするあたりをダンプすると

```
08  41  42  00  00  00  00  00  00  00  00
            ↑
```

となっていると思います。VARPTR関数で返された値は矢印の番地でした。

引数が文字型ならストリング・ディスクリプタの先頭（文字列長の格納されているところ）を返します。これは単純変数でも配列変数でも同じで，配列なら宣言された範囲で任意の添字を指定できます。ただし，値が代入されていない変数を引数にするとIllegal function call エラーになります。

各型の変数について，VARPTR関数で得られる番地以下のメモリが何を表わしているかを**図5-12**に示します。

また，引数として＃をつけたファイル番号を与えるとき，MAXFILES命令で指定されたファイル番号の最大値を超えるとBad file number エラーになります。

図5-12

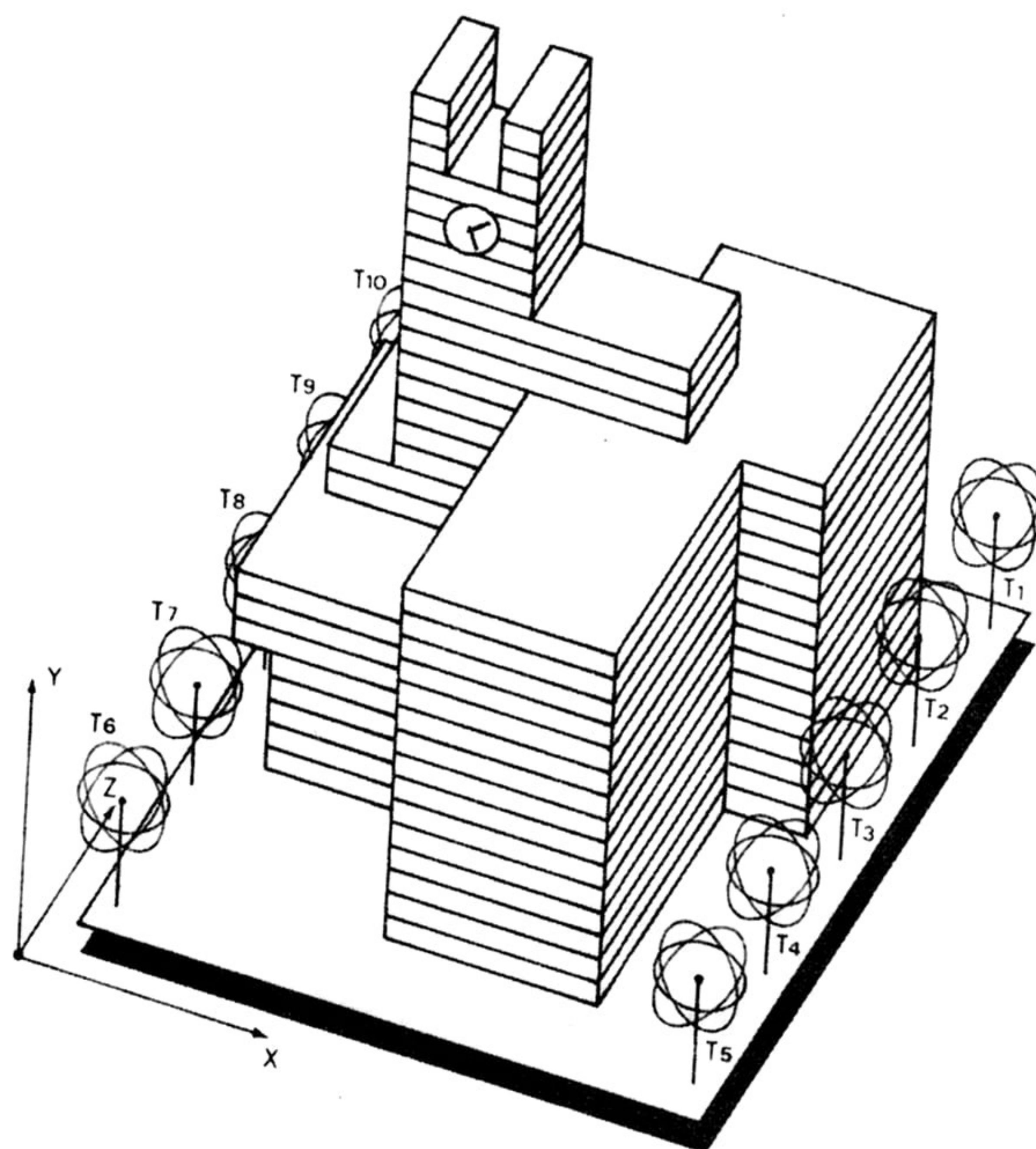
E：指数部
M：仮数部
※：VARPTRで指されるアドレス

型	※								説明
整数型	下位	上位							2バイト(値は16進で表現)
単精度型	E	M	M	M					4バイト(値はBCD表現)
倍精度型	E	M	M	M	M	M	M	M	8バイト(値はBCD表現)
文字型	文字数	下位	上位						3バイト(ストリング・ディスクリプタ)

5. PEEK,POKE (VPEEK,VPOKE)命令

PEEK，POKE命令は，BASICからメイン・メモリを直接読み書きする命令です。どちらも1命令で1バイトだけ読み書きできます。MSXでは0～7FFFH番地まではBASICインタープリタが入ったROMですから，POKE（書き込み）はできません。PEEKは全メモリ（もちろん実装されている範囲）で可能です。

VPPEK, VPOKE 命令は, VRAM に関して PEEK, POKE する命令です。MSX の VRAM は CPU (Z80) とは切り離されていて, 画面の制御を受け持つ VDP という IC が完全に管理しています。VRAM には MSX から出力する画面に関する情報のすべてが書き込まれていて, ヘタにユーザーが VPOKE すると画面を破壊する恐れがあります。

VDP や VRAM の構造に関しての詳細は本書のシリーズの「MSX fan シリーズ② MSX マシン語入門・応用編」に書かれています。興味のあるかたはそちらをごらんください。

つなげてしまおうBASICとマシン語

2 フックについて

MSXのワーク・エリアのうち，FD9AH番地からFFC9H番地まではフックのための場所です。フックはBASICの機能を拡張するために使うもので，MSXに他のコンソールやディスクなどを接続するために使います。

通常，フックにはRET命令が入っていて，BASICインタープリタは1文字入力，出力，エラー表示など多くの機能を実行する際にそれぞれに割り当てられたフック・アドレスをコールします。ふつうはRETが書かれているのですから，コールしてもすぐリターンしてきます。

そこでRET命令の代わりにジャンプ命令を入れておき，その飛び先に何らかの働きをするサブルーチンを用意しておけば，BASICの機能を拡張することができるわけです。

ここで，ちょっといたずらをしてみましょう。FEFDH番地からの3バイトのフックは，エラー表示のためのものです。これをいじって，エラーを表示するたびに“マタ　ヤッチャッタ”と表示させます。

プログラムはP158の**リスト5-5**のようになりました。HLにフックのアドレスを入れ，DEにサブルーチンMAT（“マタ　ヤッチャッタ”表示ルーチン）のアドレスを入れます。LD（HL),0C3Hでジャンプ命令（JP）を書き込み，続けてEレジスタ，Dレジスタを書き込みます。これでエラー表示のフックがMATに接続しました。

エラーが発生すると，MATではAレジスタとHLレジスタをスタックに退避してから,“マタ　ヤッチャッタ”と表示します。AレジスタとHLレジスタをもとに戻してリターンすると，あとはBASICインタープリタが“Syntax error”とか“Undefined line number”とかのメッセージを表示します。

このようにフックをいじると，いろいろとおもしろいことができます。しかし注意しないとキー入力ができなくなったり，画面に何も表示しなくなったり暴走したりします。そんなときは「マタ　ヤッチャッタ」とつぶやいて電源を切りましょう。また最初からやり直せばいいのです。そんなに落ちこまないでください。

リスト5-5

```
 MSX Self Assembler  Rev 1.0     PAGE      1


100:    00A2 =         CHPUT      EQU      00A2H
110:    FEFD =         HERRP      EQU      0FEFDH
120:
130:    D400                      ORG      0D400H
140:    D400 21FDFE               LD       HL,HERRP
150:    D403 110DD4               LD       DE,MAT
160:    D406 36C3                 LD       (HL),0C3H
170:    D408 23                   INC      HL
180:    D409 73                   LD       (HL),E
190:    D40A 23                   INC      HL
200:    D40B 72                   LD       (HL),D
210:    D40C C9                   RET
220:
230:    D40D F5        MAT:       PUSH     AF
240:    D40E E5                   PUSH     HL
250:    D40F 211FD4               LD       HL,MSG
260:    D412 7E        LOOP:      LD       A,(HL)
270:    D413 23                   INC      HL
280:    D414 B7                   OR       A
290:    D415 2805                 JR       Z,EXIT
300:    D417 CDA200               CALL     CHPUT
310:    D41A 18F6                 JR       LOOP
320:    D41C E1        EXIT:      POP      HL
330:    D41D F1                   POP      AF
340:    D41E C9                   RET
350:
360:    D41F CFC020D4 MSG:        DEFM     'ﾏﾀ ﾔｯﾁｬｯﾀ'
370:    D428 0D0A00               DEFB     0DH,0AH,0
```

つなげてしまおうBASICとマシン語

3 割り込み

MSXでは1／60秒ごとに割り込みが発生し，そのたびごとに戻り番地をスタックに入れて0038H番地をコールします（RST 38H）。そして0038H番地では全部のレジスタをスタックにしまってFD9FH番地のフックを見，割り込みの処理を実行してもとの処理へ戻ります。

したがって，ユーザーが割り込みを使用するためにはFD9FH番地にユーザーの割り込み処理ルーチンへジャンプする命令を書いておけばよいことになります。

注意しなければならないことは，ユーザーの割り込み処理ルーチンではレジスタの退避やEI命令(割り込み許可命令）実行の必要がないということです。これらはROM内のルーチンで処理してくれるわけです。

では，実際に割り込みを利用するプログラムをつくってみましょう。題して「無限ループ脱出ルーチン」。無限ループというのは，

```
LOOP:    JP    LOOP
```

のようにいつまでもぐるぐると同じところを回ってしまうループのことです。

このルーチンでは1／60秒ごとにCRTL-STOPキーが押されているかどうかをチェックし，押されていればBASICのホット・スタートへジャンプするプログラムです。プログラムは**リスト 5-6** のようになりました。まず，フックのFD6FH番地からCHKKEYへジャンプする命令を書き込み，割り込みを可能にします。これで1／60秒ごとにCHKKEYルーチンを実行するようになります。

CHKKEYルーチンではCTRL-STOPが押されたかどうかを判断するBREAKXルーチン（BIOS)をコールして，押されていればBASICのホット・スタートへジャンプします。

これで無限ループを実行しても心配いりません。CTRLキーとSTOPキーを同時に押せばBASICに戻ってくれます。

リスト5-6

```
 MSX Self Assembler  Rev 1.0     PAGE      1


100:     00A2 =          CHPUT      EQU       00A2H
110:     00B7 =          BREAKX     EQU       00B7H
120:     4167 =          HOT        EQU       4167H
130:     FD9F =          HTIMI      EQU       0FD9FH
140:
150:     D400                       ORG       0D400H
160:     D400 ED7328D4              LD        (SAVE),SP
170:     D404 2111D4                LD        HL,CHKKEY
180:     D407 22A0FD                LD        (HTIMI+1),HL
190:     D40A 3EC3                  LD        A,0C3H
200:     D40C 329FFD                LD        (HTIMI),A
210:     D40F FB                    EI
220:     D410 C9                    RET
230:
240:     D411 CDB700     CHKKEY:    CALL      BREAKX
250:     D414 3011                  JR        NC,EXIT
260:     D416 3E0A                  LD        A,0AH
270:     D418 CDA200                CALL      CHPUT
280:     D41B 3E0D                  LD        A,0DH
290:     D41D CDA200                CALL      CHPUT
300:     D420 ED7B28D4              LD        SP,(SAVE)
310:     D424 C36741                JP        HOT
320:     D427 C9         EXIT:      RET
330:
340:     D428            SAVE:      DEFS      2
```

6章 こんなこともできちゃう ランダム・テクニック

おもしろくて役に立つプログラムの数々を，学んだマシン語のテクニックで実現してしまおう。キミがほしいと思ったその機能を実現させるためのノウハウが，ここには詰まっている。あとはアイデア次第。短くておもしろいプログラムをもっと考えてみよう。

こんなこともできちゃうランダム・テクニック

1 TRONの行番号をプリンタに出す(LTRON)

BASICのデバッグをするためのコマンドにTRONという命令があります。実行している行番号を刻々と画面に表示するコマンドです。

ところが，プログラム中でLOCATEやSCREEN命令を実行すると，表示される行番号が見えなくなってTRONは役に立たなくなってしまいます。

そこでLTRON，すなわち実行している行番号をプリンタに出力するプログラムをつくりました。

このときワーク・エリア上にTRONのフックがあれば話は簡単なのですが，捜しても見つかりませんでした。代わりに，1ステートメント実行後にいつもフックする番地を見つけました。また，RUNしたときのフック，実行が終了したときのフックも使用します。

1ステートメント実行後，現在の行番号をプリンタに出力します。そして次のステートメントを実行します。このとき，直前と同じ行番号なら出力しません。

また初期設定として，RUNしたときトレース・フラグFLG（RUNコマンドによってプログラムが実行中であることを示すためのフラグ）に1を入れてトレース・モードにします。ついでに現在の行番号を65535（行番号として取りうる最大値）にしておきます。

対象プログラムの実行が終了したらFLGを0にします。

プログラムは**リスト6-1**のようになります。

リスト6-1

```
  MSX Self Assembler  Rev 1.0

1000:   00A5 =          LPTOUT    EQU     00A5H
1010:   F41C =          CURLIN    EQU     0F41CH
1020:   F672 =          MEMSIZ    EQU     0F672H
1030:   FDDB =          HPINL     EQU     0FDDBH
1040:   FECB =          HRUNC     EQU     0FECBH
1050:   FF43 =          HGONE     EQU     0FF43H
1060:
1070:   D400                      ORG     0D400H
```

```
1080:    D400 21CBFE   TRON:     LD      HL,HRUNC
1090:    D403 114BD4             LD      DE,RUN
1100:    D406 CD39D4             CALL    HOCK
1110:    D409 21DBFD             LD      HL,HPINL
1120:    D40C 1122D4             LD      DE,FLGOFF
1130:    D40F CD39D4             CALL    HOCK
1140:    D412 2143FF             LD      HL,HGONE
1150:    D415 1160D4             LD      DE,LTRON
1160:    D418 CD39D4             CALL    HOCK
1170:    D41B C9                 RET
1180:
1190:    D41C 3EC9     TROFF:    LD      A,0C9H
1200:    D41E 3243FF             LD      (HGONE),A
1210:    D421 C9                 RET
1220:
1230:    D422 F5       FLGOFF:   PUSH    AF
1240:    D423 3AD5D4             LD      A,(FLG)
1250:    D426 B7                 OR      A
1260:    D427 280E               JR      Z,FLGEXT
1270:    D429 3AD6D4             LD      A,(POS)
1280:    D42C FEFE               CP      254
1290:    D42E 2807               JR      Z,FLGEXT
1300:    D430 CD40D4             CALL    CRLF
1310:    D433 AF                 XOR     A
1320:    D434 32D5D4             LD      (FLG),A
1330:    D437 F1       FLGEXT:   POP     AF
1340:    D438 C9                 RET
1350:
1360:    D439 36C3     HOCK:     LD      (HL),0C3H
1370:    D43B 23                 INC     HL
1380:    D43C 73                 LD      (HL),E
1390:    D43D 23                 INC     HL
1400:    D43E 72                 LD      (HL),D
1410:    D43F C9                 RET
1420:
1430:    D440 3E0D     CRLF:     LD      A,0DH
1440:    D442 CDA500             CALL    LPTOUT
1450:    D445 3E0A               LD      A,0AH
1460:    D447 CDA500             CALL    LPTOUT
1470:    D44A C9                 RET
1480:
1490:    D44B F5       RUN:      PUSH    AF
1500:    D44C E5                 PUSH    HL
1510:    D44D 21FFFF             LD      HL,0FFFFH
1520:    D450 22D3D4             LD      (LINE),HL
1530:    D453 3EFE               LD      A,254
1540:    D455 32D6D4             LD      (POS),A
1550:    D458 3E01               LD      A,1
```

```
1560:   D45A 32D5D4              LD      (FLG),A
1570:   D45D E1                  POP     HL
1580:   D45E F1                  POP     AF
1590:   D45F C9                  RET
1600:
1610:   D460 F5       LTRON:     PUSH    AF
1620:   D461 3AD5D4              LD      A,(FLG)
1630:   D464 B7                  OR      A
1640:   D465 2810                JR      Z,EXIT
1650:   D467 E5                  PUSH    HL
1660:   D468 D5                  PUSH    DE
1670:   D469 2AD3D4              LD      HL,(LINE)
1680:   D46C ED5B1CF4            LD      DE,(CURLIN)
1690:   D470 ED52                SBC     HL,DE
1700:   D472 C479D4              CALL    NZ,PRINT
1710:   D475 D1                  POP     DE
1720:   D476 E1                  POP     HL
1730:   D477 F1       EXIT:      POP     AF
1740:   D478 C9                  RET
1750:
1760:   D479 C5       PRINT:     PUSH    BC
1770:   D47A 3AD6D4              LD      A,(POS)
1780:   D47D 3C                  INC     A
1790:   D47E 32D6D4              LD      (POS),A
1800:   D481 FE0B                CP      11
1810:   D483 3808                JR      C,PRTSKP
1820:   D485 CD40D4              CALL    CRLF
1830:   D488 3E01                LD      A,1
1840:   D48A 32D6D4              LD      (POS),A
1850:   D48D 3E20     PRTSKP:    LD      A,' '
1860:   D48F CDA500              CALL    LPTOUT
1870:   D492 011027              LD      BC,10000
1880:   D495 EB                  EX      DE,HL
1890:   D496 EB       LOOP:      EX      DE,HL
1900:   D497 CDB9D4              CALL    DIV
1910:   D49A 7B                  LD      A,E
1920:   D49B C630                ADD     A,30H
1930:   D49D CDA500              CALL    LPTOUT
1940:   D4A0 79                  LD      A,C
1950:   D4A1 3D                  DEC     A
1960:   D4A2 280E                JR      Z,PRTEXT
1970:   D4A4 59                  LD      E,C
1980:   D4A5 50                  LD      D,B
1990:   D4A6 010A00              LD      BC,10
2000:   D4A9 E5                  PUSH    HL
2010:   D4AA CDB9D4              CALL    DIV
2020:   D4AD E1                  POP     HL
2030:   D4AE 4B                  LD      C,E
```

```
2040:    D4AF 42                   LD      B,D
2050:    D4B0 18E4                 JR      LOOP
2060:    D4B2 3E20     PRTEXT:     LD      A,' '
2070:    D4B4 CDA500               CALL    LPTOUT
2080:    D4B7 C1                   POP     BC
2090:    D4B8 C9                   RET
2100:
2110:    D4B9 210000   DIV:        LD      HL,0
2120:    D4BC D9                   EXX
2130:    D4BD 0610                 LD      B,16
2140:    D4BF D9       DIVLP:      EXX
2150:    D4C0 CB23                 SLA     E
2160:    D4C2 CB12                 RL      D
2170:    D4C4 ED6A                 ADC     HL,HL
2180:    D4C6 ED42                 SBC     HL,BC
2190:    D4C8 3803                 JR      C,SKIP
2200:    D4CA 1C                   INC     E
2210:    D4CB 1801                 JR      SKIP1
2220:    D4CD 09       SKIP:       ADD     HL,BC
2230:    D4CE D9       SKIP1:      EXX
2240:    D4CF 10EE                 DJNZ    DIVLP
2250:    D4D1 D9                   EXX
2260:    D4D2 C9                   RET
2270:    D4D3          LINE:       DEFS    2
2280:    D4D5          FLG:        DEFS    1
2290:    D4D6          POS:        DEFS    1
```


2 リストを見られないようにする(UNLIST)

UNLISTとは，LISTコマンドを実行してもプログラム・リストを表示しないようにしてしまうプログラムのことです。

原理は簡単。LISTコマンドを実行するときに，スクリーン・モードをSCREEN 2にしてしまうのです。リストやエラーメッセージなどの文字は必ずスクリーン・モード0または1で表示されるので，こうして強制的にスクリーン・モード2にしてしまえば人間には見られないのです。

プログラムは**リスト 6-2** のようになります。LISTコマンドのフックにSCREEN　2へのジャンプ命令を書き込むだけです。

リスト6－2

```
 MSX Self Assembler  Rev 1.0    PAGE    1


100:    0072 =          INIGRP    EQU     0072H
110:    FF89 =          HLIST     EQU     0FF89H
120:                    ;
130:    D400                      ORG     0D400H
140:    D400 2189FF               LD      HL,HLIST
150:    D403 117200               LD      DE,INIGRP
160:    D406 36C3                 LD      (HL),0C3H
170:    D408 23                   INC     HL
180:    D409 73                   LD      (HL),E
190:    D40A 23                   INC     HL
200:    D40B 72                   LD      (HL),D
210:    D40C C9                   RET
```

3 こんなこともできちゃうランダム・テクニック NEWしてしまったプログラムを復活させる

「あっ，しまった。セーブする前にNEWしちゃった」

こんな経験はありませんか。せっかく入力したのにもう1度入力なんてうんざりです。なんとかできないものでしょうか。

実はNEWコマンドを実行しても，プログラムはほとんどメモリ中に残っているのです。NEWコマンドはメモリ中の数バイトを書きかえるだけで，そこを戻せばプログラムももと通りになります。そのためにはBASICのテキストがどのようにメモリ中に格納されているかを知る必要があります。第5章の1.を読んでください。

リスト 6-3を見てください。このプログラムは**図 6-1** のようにメモリに格納されています（16 KRAMシステムでは8000H番地からではなく，C000H番地から格納されている)。

リスト6－3

```
10 A=0
20 PRINT A
```

図6－1

```
8000 00 09 80 0A 00 41 EF 11 : D4
8008 00 11 80 14 00 91 20 41 : 97
8010 00 00 00 14 FF 00 FF 00 : 12
8018 FF 00 FF 00 FF 00 FF 00 : FC
```

ここでNEWコマンドを実行してからもう1度メモリの中を見てみると **図 6-2** のようになります。すなわち，8001H番地と8002H番地がゼロになっています。ここをモニタでもとに戻してみてください。NEWしたはずのプログラム・リストを見ることができます。

図 6-2

```
8000 00 00 00 0A 00 41 EF 11 : 4B
8008 00 11 80 14 00 91 20 41 : 97
8010 00 00 00 14 FF 00 FF 00 : 12
8018 FF 00 FF 00 FF 00 FF 00 : FC
```

これと同じことをするプログラムをつくれば，NEWしてしまったプログラムを復活させることができます。プログラムは **リスト 6-4** のようになります。まず最初の行の終わりを示すゼロを探します。このとき注意しなければならないのは，最初の行で&HやGOTO命令を使っているときです。

たとえば　B=&H2 というプログラムは，

42　EF　0C　02　00

という内部表現でメモリに格納されます。42が“B”のASCIIコード，EFが“=”の中間コード，0C，02，00が“&H2”を表わしています。ここに現われる最後のゼロは行の終わりを示すものではなく，16進数0002の上位バイトを表わしているものです。0Cが「続く2バイトは16進数だ」ということを示す識別コードです。このようなときは2バイトを無視してゼロを探せばよいことになります。その処理をしているのが190行～350行です。

さて，行の最後が見つかったらそれを8001H番地と8002H番地（16Kのときは C001H と C002H 番地）に格納します。そしてプログラムの終わりを探し，これをワーク・エリアのVARTAB，ARYTAB，STRENDに格納します。これらの番地の意味は順に単純変数領域開始番地，配列変数領域開始番地，使用中のメモリの最後の番地です（第5章の1.参照）。

リスト 6-4

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    F676 =          TXTTAB      EQU      0F676H
110:    F6C2 =          VARTAB      EQU      0F6C2H
120:    F6C4 =          ARYTAB      EQU      0F6C4H
```

```
130:    F6C6 =          STREND      EQU      0F6C6H
140:
150:    D400                        ORG      0D400H
160:    D400 2A76F6                 LD       HL,(TXTTAB)
170:    D403 010400                 LD       BC,4
180:    D406 09                     ADD      HL,BC
190:    D407 7E         LOOP:       LD       A,(HL)
200:    D408 23                     INC      HL
210:    D409 B7                     OR       A
220:    D40A 2815                   JR       Z,FIND
230:    D40C D60B                   SUB      0BH
240:    D40E 38F7                   JR       C,LOOP
250:    D410 FE15                   CP       20H-0BH
260:    D412 30F3                   JR       NC,LOOP
270:    D414 E5                     PUSH     HL
280:    D415 213BD4                 LD       HL,TBL
290:    D418 4F                     LD       C,A
300:    D419 0600                   LD       B,0
310:    D41B 09                     ADD      HL,BC
320:    D41C 4E                     LD       C,(HL)
330:    D41D E1                     POP      HL
340:    D41E 09                     ADD      HL,BC
350:    D41F 18E6                   JR       LOOP
360:    D421 EB         FIND:       EX       DE,HL
370:    D422 2A76F6                 LD       HL,(TXTTAB)
380:    D425 73                     LD       (HL),E
390:    D426 23                     INC      HL
400:    D427 72                     LD       (HL),D
410:    D428 EB         LOOP2:      EX       DE,HL
420:    D429 5E                     LD       E,(HL)
430:    D42A 23                     INC      HL
440:    D42B 56                     LD       D,(HL)
450:    D42C 7A                     LD       A,D
460:    D42D B3                     OR       E
470:    D42E 20F8                   JR       NZ,LOOP2
480:    D430 23                     INC      HL
490:    D431 22C2F6                 LD       (VARTAB),HL
500:    D434 22C4F6                 LD       (ARYTAB),HL
510:    D437 22C6F6                 LD       (STREND),HL
520:    D43A C9                     RET
530:
540:    D43B 02020202 TBL:          DEFB     2,2,2,2
550:    D43F 01000000               DEFB     1,0,0,0
560:    D443 00000000               DEFB     0,0,0,0
570:    D447 00000000               DEFB     0,0,0,0
580:    D44B 00020400               DEFB     0,2,4,0
590:    D44F 08                     DEFB     8
```

4 こんなこともできちゃうランダム・テクニック

日本語エラーメッセージ

最後に，フックを使ってエラーメッセージを日本語で表示してみましょう。

リスト 6-5を見てください。まずエラー表示のフックに JERR 番地へのジャンプ命令を書き込みます。これでエラーが発生すると JERR 番地へ処理が移ります。このとき，C レジスタにエラー番号が入っているので，その分だけ．（ピリオド）の数をかぞえて（余分なメッセージを飛ばして）エラー番号に対応するメッセージを出力します。

このプログラムは

CLEAR 300, &HD200 ↵

としてからアセンブルし，D200H 番地から実行します。1度実行した後は，エラーが起きたとき日本語と英語の両方でエラーを教えてくれます。

リスト 6-5

```
 MSX Self Assembler  Rev 1.0     PAGE     1


100:    00A2 =         CHPUT       EQU      00A2H
110:    FEFD =         HERRP       EQU      0FEFDH
120:
130:    D200                       ORG      0D200H
140:    D200 110DD2                LD       DE,JERR
150:    D203 21FDFE                LD       HL,HERRP
160:    D206 36C3                  LD       (HL),0C3H
170:    D208 23                    INC      HL
180:    D209 73                    LD       (HL),E
190:    D20A 23                    INC      HL
200:    D20B 72                    LD       (HL),D
210:    D20C C9                    RET
220:
230:    D20D E5        JERR:       PUSH     HL
240:    D20E F5                    PUSH     AF
250:    D20F C5                    PUSH     BC
260:    D210 213DD2                LD       HL,ERRM
270:    D213 7E        REPEAT:     LD       A,(HL)
280:    D214 23                    INC      HL
290:    D215 FEFF                  CP       0FFH
300:    D217 2816                  JR       Z,EXIT
310:    D219 FE2E                  CP       '.'
320:    D21B 20F6                  JR       NZ,REPEAT
330:    D21D 0D                    DEC      C
340:    D21E 20F3                  JR       NZ,REPEAT
350:    D220 7E        LOOP:       LD       A,(HL)
360:    D221 23                    INC      HL
370:    D222 FEFF                  CP       0FFH
380:    D224 2809                  JR       Z,EXIT
390:    D226 FE2E                  CP       '.'
400:    D228 2805                  JR       Z,EXIT
```

```
410:    D22A CDA200               CALL    CHPUT
420:    D22D 18F1                 JR      LOOP
430:    D22F 3E0D      EXIT:      LD      A,0DH
440:    D231 CDA200               CALL    CHPUT
450:    D234 3E0A                 LD      A,0AH
460:    D236 CDA200               CALL    CHPUT
470:    D239 C1                   POP     BC
480:    D23A F1                   POP     AF
490:    D23B E1                   POP     HL
500:    D23C C9                   RET
510:
520:    D23D 2E        ERRM:      DEFM    '.'
530:    D23E 464F5220             DEFM    'FOR ｶﾞ ｱﾘﾏｾﾝ.'
540:    D24B C6ADB3D8             DEFM    'ﾆｭｳﾘｮｸ ﾏﾁｶﾞｲ.'
550:    D258 474F5355             DEFM    'GOSUB ｶﾞ ｱﾘﾏｾﾝ.'
560:    D267 C3DEB0C0             DEFM    'ﾃﾞｰﾀ ｶﾞ ﾀﾘﾏｾﾝ.'
570:    D275 B6DDBDB3             DEFM    'ｶﾝｽｳ ﾉ ﾋｷｽｳ ｶﾞ ﾏﾁｶﾞｯﾃ ｲﾏｽ.'
580:    D28F B9AFB620             DEFM    'ｹｯｶ ｶﾞ ｵｵｷｽｷﾞﾏｽ.'
590:    D29F D2D3D820             DEFM    'ﾒﾓﾘ ｶﾞ ﾀﾘﾏｾﾝ.'
600:    D2AC BFC920B7             DEFM    'ｿﾉ ｷﾞｮｳ ﾊ ｱﾘﾏｾﾝ.'
610:    D2BC CAB2DAC2             DEFM    'ﾊｲﾚﾂ ﾉ ｿｴｼﾞ ｶﾞ ｵｵｷｽｷﾞﾏｽ.'
620:    D2D4 B5C5BCDE             DEFM    'ｵﾅｼﾞ ﾊｲﾚﾂ ｦ ﾃｲｷﾞ ｼﾃｲﾏｽ.'
630:    D2EB BEDEDB20             DEFM    'ｾﾞﾛ ﾃﾞ ﾜﾘｻﾞﾝ ｼﾏｼﾀ.'
640:    D2FD BFC920CC             DEFM    'ｿﾉ ﾌﾞﾝ ﾊ ﾀﾞｲﾚｸﾄ ﾓｰﾄﾞ ﾃﾞﾊ ﾂｶｴﾏｾﾝ.'
650:    D31D CDDDBDB3             DEFM    'ﾍﾝｽｳ ﾉ ｶﾀ ｶﾞ ﾁｶﾞｲﾏｽ..'
660:    D331 D3BCDECD             DEFM    'ﾓｼﾞﾍﾝｽｳ ﾉ ｴﾘｱ ｶﾞ ﾀﾘﾏｾﾝ.'
670:    D348 D3BCDEDA             DEFM    'ﾓｼﾞﾚﾂ ﾉ ﾅｶﾞｻ ｶﾞ 255 ｦ ｺｴﾏｼﾀ.'
680:    D364 D3BCDEBC             DEFM    'ﾓｼﾞｼｷ ｶﾞ ﾌｸｻﾞﾂ ｽｷﾞﾏｽ.'
690:    D379 434F4E54             DEFM    'CONT ﾒｲﾚｲ ﾊ ﾂｶｴﾏｾﾝ.'
700:    D38C D5B0BBDE             DEFM    'ﾕｰｻﾞｰ ｶﾝｽｳ ｶﾞ ﾃｲｷﾞ ｻﾚﾃｲﾏｾﾝ.'
710:    D3A7 B6BEAFC4             DEFM    'ｶｾｯﾄ ﾄﾉ ﾆｭｳｼｭﾂﾘｮｸ ｶﾞ ﾁｭｳﾀﾞﾝ ｼﾏｼﾀ.'
720:    D3C8 C0C0DEBC             DEFM    'ﾀﾀﾞｼｸ ｾｰﾌﾞ ｻﾚﾃｲﾏｾﾝ.'
730:    D3DB 52455355             DEFM    'RESUME ｶﾞ ｱﾘﾏｾﾝ.'
740:    D3EB B4D7B020             DEFM    'ｴﾗｰ ﾃﾞ ﾅｲﾉﾃﾞ RESUME ﾃﾞｷﾏｾﾝ.'
750:    D406 C3B2B7DE             DEFM    'ﾃｲｷﾞ ｻﾚﾃｲﾅｲ ｴﾗｰ ｶﾞ ﾊｯｾｲ ｼﾏｼﾀ.'
760:    D423 B5CDDFD7             DEFM    'ｵﾍﾟﾗﾝﾄﾞ ｶﾞ ﾀﾘﾏｾﾝ.'
770:    D434 FF                   DEFB    0FFH
```

あとがき

これは「BASICにもあるんだから，マシン語にだって定石があるはずだ」こんなことを思っているあなたのための本です。マシン語で意味のある動作をさせるために，ロード命令やコール，ジャンプ命令をどのように組み合わせればよいか。ビギナーがおちいりやすい落とし穴は何か。実際のプログラミングでよく登場する命令の並び（定石）はどんなものか。実際にプログラムを作成する立場に自らを置き，ビギナーが実践しながらマシン語プログラミングを習得できるよう構成しました。公開されているリストもニモニックが表示されるアセンブル・リスト形式になっていますから，その構造や手順，テクニックがひと目で理解できたと思います。もちろん改造も自由自在です。

本書はMSXユーザーならずとも，Z80 CPUを搭載したマシンのユーザーなら必ず参考になる内容と自負しています。本書を1つのステップに，マシン語を身近にしてください。

〈参考文献〉

（1）MSX早わかり事典　朝日新聞社
（2）ASCII・1983年 8月号，10月号　アスキー
（3）MSXマシン語入門・基礎編　MIA
（4）BASIC to ASSEMBLER　CQ出版社
（5）Z-80 マイクロコンピュータ　丸善
（6）PC-Techknow 8000 Vol.1.1　アスキー
（7）PC-Techknow 6000 Vol.1.1　アスキー
（8）8080／Z-80 アセンブリ言語　近代科学社
（9）MSXユーザーズマニュアル

MSX INFORMATION

発売中

MSX FANシリーズ マシン語入門（基礎編）

ＭＳＸでマシン語を学ぶ人のために、予備知識、基礎知識からマシン語プログラミングの実際まで、図表等を多用してわかりやすく解説。

〈内容〉マシン語のための予備知識●マシン語の長所と短所●マシン語の表わしかた／基礎知識●コンピュータの構造●マイコンで扱う数●２進演算／Z80Aのマシン語命令●Z80AのCPUについて／Z80Aのニモニックとマシン語／モニタ・アセンブラ／マシン語プログラムの作成方法

B5判160頁　定価1800円（〒250円）

続刊

MSX FANシリーズ マシン語入門（応用編）

マシン語ゲームづくりに必要なハード的な知識（特に表示、音）を、サンプル・プログラムと図表を多用して徹底解説。グラフィック・エディタ、サウンド・コンパイラ等のツール・ソフトも充実。ＭＳＸ音声合成（ＭＳＸがしゃべる！）にも注目を。FANシリーズ①『マシン語入門（基礎編）』に続く待望の第２弾。

〈内容〉ゲーム制作のプレリュード／グラフィック機能を使いこなそう●VDPの機能●表示モード●VDPのレジスタ●スプライト機能他。

B５判　定価1800円（〒250円）

新発売

MSX マシン語ゲーム集

①おてんばベッキーの大冒険②ファイナル麻雀③ニョロルス④アドベン・チュー太⑤ジャンピング・ラビット⑥ジグソーセット⑦ロンサム・タンク進撃

ゲーム7本　A5判
定価1500円（〒250円）

MSX FANシリーズ3　**マシン語入門〔実践編〕**

1984年8月1日

著　者　渡辺卓也　樋口賢治
編集人　MIA第2編集部
発行人　塚本慶一郎
発行所　株式会社 エム・アイ・エー
〒102 東京都千代田区紀尾井町3-20
電話(03)265-2461(代)
イラスト　大村敦郎
印刷・製本　株式会社 詩文字美巧印刷

ISBN4-87170-021-6　C3055　¥1800E

アンケート

本誌をお買いあげいただきありがとうございました。みなさんのご意見を編集の参考にさせていただきたいと思います。アンケートにご協力ください。

1 本誌を何でお知りになりましたか。

●書店〔　　〕 ●広告〔　　〕 ●友人，知人〔　　〕
●パソコンショップ〔　　〕 ●その他〔　　　　　〕
●店名〔　　　　　　　　　　〕

2 本誌をどこでお買いになりましたか。

●書店〔　　〕 ●パソコンショップ〔　　〕
●その他〔　　　　　　　　　　〕
●店名〔　　　　　　　　　　〕

3 本誌の内容はいかがでしたか。感想をお書きください。

〔　　　　　　　　　　　　　　　　　〕

4 本誌内容の編集部へのご意見・ご希望をお聞かせください。

〔　　　　　　　　　　　　　　　　　〕

5 今後とりあげてほしい企画がありましたらお書きください。

〔　　　　　　　　　　　　　　　　　〕

郵便はがき

お手数ですが
40円
切手をお貼りください。

102-□□

東京都千代田区紀尾井町
三の二〇

(株)エム・アイ・エー
マシン語入門(実践編)
編集部係

きりとり線

フリガナ		性別 男・女
氏名		年令 才
ご住所	〒 Tel.( )	
勤務先・所属部課 学校名・学科名		
お持ちのマイコン		使用する プログラム言語
マイコン・クラブ		経験 年

MIA COMPUTER BOOKS

MSX fan series

MACHINE LANGUAGE

MIA

MICRO INFORMATION ASSOCIATES

ISBN4-87170-021-6 C3055 ¥1800E

定価1800円